

DTIC FILE COPY

RADC-TR-90-140
Final Technical Report
July 1990



②

AD-A226 698

DISTRIBUTED SYSTEM MODELING ENVIRONMENT (DSME)

PAR Government Systems Corporation

C.L. Brown and F.K. Frantz

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

DTIC
ELECTE
SEP 19 1990
S E D

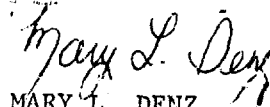
Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

90 09 18 210

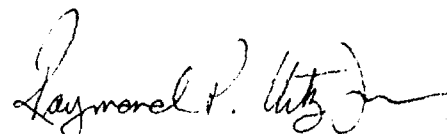
This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS) At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-90-140 has been reviewed and is approved for publication.


APPROVED:


MARY L. DENZ
Project Engineer

APPROVED:


RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command and Control

FOR THE COMMANDER:


IGOR G. PLONISCH
Directorate of Plans and Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTD) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of Management and Budget, Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE July 1990	3. REPORT TYPE AND DATES COVERED Final Aug 86 - Mar 90	
4. TITLE AND SUBTITLE DISTRIBUTED SYSTEM MODELING ENVIRONMENT (DSME)			5. FUNDING NUMBERS C - F30602-86-C-0195 PE - 63278F PR - 2530 TA - 01 WU - 27	
6. AUTHOR(S) C. L. Brown and F. K. Frantz				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) PAR Government Systems Corporation 22 Seneca Turnpike New Hartford NY 13413			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Air Development Center (COTD) Griffiss AFB NY 13441-5700			10. SPONSORING/MONITORING AGENCY REPORT NUMBER RADC-TR-90-140	
11. SUPPLEMENTARY NOTES RADC Project Engineer: Mary L. Denz/COTD/(315) 330-3623				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This report documents the activities and results of the Distributed Systems Modeling Environment (DSME) contract (F30602-86-C-0195) performed by PAR Government Systems Corporation (PGSC) and Harris Corporation for the Rome Air Development Center (RADC). The DSME is a computer-based environment that will assist the development and analysis of distributed computer systems by providing support for the simulation modeling of these systems and their components. During the development of the overall DSME concept, PGSC/Harris and RADC defined an overall context for the proposed DSME and defined the framework of that system. In addition, the particular goals of the current effort were established in relation to the overall target system, which is expected to evolve through long-term development. The current effort focuses on the initial design and implementation of a limited prototype demonstration of the DSME concept.				
14. SUBJECT TERMS Model Distributed System Simulation Experiment Environment System Under Study			15. NUMBER OF PAGES 164	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED			18. PRICE CODE	
18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED		19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED		20. LIMITATION OF ABSTRACT SAR

Contents

1.	Introduction	1-1
1.1	Contract Summary	1-1
1.2	Summary of DSME Background	1-2
1.3	Summary of DSME Requirements	1-3
1.4	Summary of Baseline DSME Design Concepts	1-5
1.5	Overview of the DSME Concept Demonstration System	1-6
1.6	Summary of Lessons Learned and Future Directions	1-8
2.	DSME Context	2-1
2.1	DSME Background	2-1
2.1.1	The Need for Distributed Computer Systems	2-1
2.1.2	Approaches to Achieving Distribution	2-3
2.1.3	RADC Distributed Systems Efforts	2-4
2.2	Development Processes Supported by DSME	2-8
2.2.1	Distributed System Development Procedures	2-9
2.2.2	Distributed System Performance Analysis Procedures	2-11
2.2.3	An Example of Simulation System Performance Analysis	2-12
2.2.4	The Experimentation Process	2-13
2.2.5	The Simulation Process	2-15
2.3	Existing Capabilities	2-18
2.3.1	Distributed System Modeling	2-18
2.3.2	System Performance Evaluation	2-19
2.3.3	Data Manipulation and Analysis	2-19
3.	DSME Requirements	3-1
3.1	General Requirements of DSME	3-1
3.1.1	Generality	3-1
3.1.2	Flexibility	3-3
3.1.3	Modularity	3-4
3.1.4	Usability	3-5
3.1.5	Validity and Testability	3-8

Contents (cont'd)

3.2	Tool Design Characteristics	3-8
3.2.1	Consistent Interface.....	3-8
3.2.2	Generic Nature.....	3-9
3.2.3	Ease of Collection.....	3-9
3.2.4	Effective Data Manipulation and Analysis Capabilities.....	3-10
3.2.5	Efficient Storage.....	3-10
3.3	Functional Requirements.....	3-10
3.3.1	Experiment Definition.....	3-10
3.3.2	Experiment Implementation.....	3-11
3.3.3	Experiment Execution.....	3-11
3.3.4	Data Manipulation and Analysis.....	3-12
3.4	Assumptions and Constraints	3-13
3.5	Baseline DSME Objectives	3-16
3.5.1	The Baseline DSME Concept	3-16
3.5.2	Simulation Environment Objectives.....	3-21
3.5.3	Distributed System Evaluation Objectives.....	3-24
4.	Baseline DSME Design Concept	4-1
4.1	Design Representation.....	4-1
4.1.1	Object-Oriented Problem Definition	4-1
4.1.2	Baseline DSME Data-Flow Representation	4-4
4.1.3	Data Dictionary.....	4-21
4.2	Architectural Considerations.....	4-25
4.2.1	General Software Configurations.....	4-25
4.2.2	General Data Collection Techniques.....	4-29
4.2.3	Experimentation Abstraction.....	4-35
4.2.4	Recommended Approach – System Services	4-40
4.3	Summary Concept of Operations.....	4-41
4.3.1	SUS Incorporation into Baseline DSME.....	4-41
4.3.2	Experiment Preparation.....	4-42
4.3.3	Experiment Execution.....	4-44
4.3.4	Experiment Analysis.....	4-45

Contents (cont'd)

4.4	Design Concept Applied to SIM DRIVER.....	4-46
4.4.1	Test Case Discussion	4-46
4.4.2	Data Collection Abstraction	4-46
4.5	Other Issues.....	4-48
4.5.1	Analysis Function Context.....	4-48
4.5.2	Analysis Implementation Potential	4-49
4.5.3	Performance Measurement Self-Interference	4-49
4.5.4	Experiment Stimulation Requirement.....	4-49
4.5.5	Use of Object Orientation	4-50
4.5.6	DSME Target Environment.....	4-50
5.	DSME Concept Demonstration System.....	5-1
5.1	Demonstration Experiment.....	5-1
5.2	Concept Demonstration System Architecture	5-2
5.3	User Interface Component.....	5-4
5.4	DSME Database.....	5-20
5.5	Experiment Preparation Component.....	5-27
5.6	System Under Study: Simulation Driver Integration.....	5-27
5.6.1	Simulation Driver Integration System Description	5-27
5.6.2	Modifications to Simulation Driver Integration	5-32
5.7	Loader Component.....	5-33
5.8	Reporter Component	5-33
6.	Lessons Learned and Future Directions	6-1
6.1	Lessons Learned.....	6-1
6.2	Future Directions	6-5
6.2.1	DSME/DISE Relationship Definition.....	6-6
6.2.2	Develop Instrumentation Tools	6-7
6.2.3	Enhanced Data Collection Facilities.....	6-7
6.2.4	COTS DBMS and Statistics Packages.....	6-8
6.2.5	Develop Concept of Abstraction	6-11
6.2.6	Another SUS	6-12
6.2.7	Further System Development.....	6-13

List of Figures

Figure No.	Title	Page No.
1-1	DSME Concept Demonstration Architecture.....	1-7
2-1	Distributed Systems Development Process.....	2-10
2-2	Distributed System Monitoring Experimentation Process	2-14
2-3	Simulation Process.....	2-16
3-1	DSME Concept.....	3-19
3-2	Potential Modeling Scopes for DSME.....	3-20
4-1	DSME DFD – DSME Top Level.....	4-5
4-2	DSME DFD – Incorporate SUS (1).....	4-6
4-3	DSME DFD – Conduct Experimentation (2).....	4-8
4-4	DSME DFD Prepare Experiment (2.1)	4-9
4-5	DSME DFD Define SUS Configuration (2.1.1).....	4-10
4-6	DSME DFD Define Experiment Analysis Requirements (2.1.2).....	4-12
4-7	DSME DFD Define Experiment Goals (2.1.2.1).....	4-13
4-8	DSME DFD Execute Experiment (2.2)	4-15
4-9	DSME DFD Configure Experiment (2.2.1).....	4-16
4-10	DSME DFD Run Experiment (2.2.2)	4-17
4-11	DSME DFD Analyze Experiment (2.3)	4-19
4-12	DSME DFD Manipulate Data (2.3.2)	4-20
4-13	DSME DFD Generate Reports (2.3.3)	4-22
4-14	Software Configuration	4-27
4-15	Collection Method Approaches	4-30
4-16	Experimentation Abstraction	4-38
5-1	DSME Concept Demonstration Architecture.....	5-3
5-2	CSCI Top Level Data Flow.....	5-6
5-3	DSME Data Structure.....	5-8
5-4	DSME Genealogy Screen.....	5-9
5-5	DSME Attribute Screen.....	5-11
5-6	DSME Relationship Screen	5-13
5-7	DSME Script Screen	5-15
5-8	Simulation Driver Integration Genealogy Structure.....	5-21
5-9	Simulation Driver Integration Attribute Example	5-23
5-10	Simulation Driver Integration Relationship Structure.....	5-24
5-11	Simulation Driver Integration Relationship Structure (Cont'd)	5-25
5-12	Simulation Driver Integration Software Architecture	5-31

List of Tables

Table No.	Title	Page No.
4-1	Collection Method Summary.....	4-29
5-1	Genealogy Screen Command Definitions.....	5-10
5-2	Attributes Screen Command Definitions.....	5-12
5-3	Relationships Screen Command Definitions.....	5-14
5-4	Scripts Screen Command Definitions.....	5-16

Accession For	
NEW CHART	<input checked="" type="checkbox"/>
DELETED	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Flat	Avail and/or Special
A-1	



1. INTRODUCTION

This *Final Technical Report* documents the activities and results of the Distributed Systems Modeling Environment (DSME) contract (number F30602-86-C-0195), performed by PAR Government Systems Corporation (PGSC) and Harris Corporation for the Rome Air Development Center (RADC). This reports fulfills the requirements of Data Item A007 of the subject contract.

This introductory section provides a synopsis of the report. Its organization parallels the organization of the entire document, beginning with some overall comments summarizing the objectives and accomplishments of the DSME effort and a discussion of the background which gave the DSME effort its impetus and defines its context. Requirements for DSME are documented in Section 1.3. Section 1.4 discusses architectural and other technical considerations; Section 1.5 describes the DSME Concept Demonstration system that was implemented under the effort; and Section 1.6 summarizes potential future directions.

1.1 CONTRACT SUMMARY

The objective of this effort, as defined in the contract Statement of Work, was "to develop a computer-based environment of system performance evaluation tools and methods capable of effectively supporting quantitative analysis of current distributed automated data processing (ADP) systems capable of tactical and strategic operational systems."¹ The initial exercise of defining the requirements for such an environment quickly led to the conclusion that complete development was well beyond the scope of the effort. A substantial amount of effort under the contract was expended to define an underlying concept on which the development of the environment could be founded.

Even the definition of a concept turned out to have some pitfalls of its own. One issue raised repeatedly throughout the contract was the role of simulation. Simulation is an important tool for supporting the development of distributed systems; at the same time, some of the performance evaluation tools were particularly useful in developing distributed simulations. The issue of setting the bounds for such an environment was also a problematic issue. However, after reaching a consensus on the underlying concept, it was possible to identify some of the key technical issues, and to develop a prototype capability to investigate and demonstrate some key aspects of a DSME.

The exercise of defining the boundaries of DSME led to the realization that a truly comprehensive distributed system development support environment involved much more than was originally planned for the DSME effort. Ideally, a design for the entire concept would have been developed; then some component of that design would have been implemented. But in fact, even the design for the entire capability would have overwhelmed contract resources. For that reason, the design and to a greater extent the subsequent

1. Rome Air Development Center, Distributed System Modeling Environment Statement of Work, PR Number B-6-3510, 4 November 1985.

implementation addressed one aspect of the DSME concept, namely the support for the simulation analysis aspect of the distributed system development process.

In particular, the Concept Demonstration system provides a vehicle for demonstrating, and experimenting with, some of the key aspects of the DSME concept: a uniform user interface, experiment definition, performance and functional data collection from a system, database support for these capabilities, and data analysis/report generation.

1.2 SUMMARY OF DSME BACKGROUND

The motivation for the DSME effort evolves from the rapid advances in both hardware capabilities and processing requirements. In particular, advances in processing capacities, communications capabilities, and network management have made large scale distributed systems feasible. On the other hand, user requirements for fault tolerant, computationally intensive systems has also grown dramatically over the past few years. While significant research has been (and is currently being) conducted to address the "components" of distributed system technology, a critical aspect that must also be addressed is a consistent and comprehensive approach to the development of distributed systems. This is the motivation for the DSME effort.

There are a number of approaches to dividing processing load among multiple processors, including vector processing, pipelined processing, and distributed processing. In order to bound the scope of the effort, and in order to focus on the area of greatest interest to RADC/COTD, a "distributed system" is defined, for the purposes of this effort, as any networked system of computers operating under a distributed operating system or operating under the control of a distributed application.

A number of efforts at RADC are addressing related aspects of distributed system technology development, including:

- Simulation tools, such as the Internetted System Modeling (ISM) system;
- Distributed operating systems, such as Cronus and Alpha;
- Distributed planning, such as the Survivable Adaptive Planning Experiment (SAPE); and
- Distributed databases.

The focus of DSME is on defining a comprehensive environment for supporting the development of distributed systems. To that end, it is necessary to understand the processes that are followed in developing a distributed system. Relevant processes include:

- Distributed system development processes;
- Distributed system performance analysis processes;
- Simulation system performance analysis processes;
- Experimentation processes; and
- Simulation processes.

A number of existing tools support various aspects of these processes. For example, there are performance monitoring tools that are typically supplied with operating systems that support some distributed system performance analysis processes. However, these tools come from a variety of sources (some are commercially available packages, some are the results of research efforts, etc.), have varying degrees of coupling to the operating systems (performance monitoring tools tend to be tightly coupled to the operating system, whereas data analysis tools tend to be independent of the operating system), and do not completely support the processes identified above. The emphasis for DSME, then, is to define an environment for distributed system development that both exploits existing capabilities and identifies additional required tools.

More detailed discussion of these background issues is contained in Section 2.

1.3 SUMMARY OF DSME REQUIREMENTS

The first step in the analysis of DSME requirements was to establish the general characteristics of the DSME system. The characteristics considered most critical to meeting the DSME objectives include:

- Generality;
- Flexibility;
- Modularity;
- Usability; and
- Validity and Testability.

Within the DSME, there is a set of tools requiring the following specific characteristics:

- Consistent interface;

- Generic nature;
- Ease of data collection;
- Effective manipulation and analysis capabilities; and
- Efficient storage.

These tools must support a number of functions which were identified as required functions for a DSME. These functions include:

1. Defining an experiment.
2. Implementing an experiment.
3. Executing an experiment.
4. Manipulating and analyzing the data.

Several constraints limit the number of design and implementation solutions that can be developed for a DSME to meet the requirements outlined above. These constraints include:

1. Other ongoing efforts, such as the Distributed System Evaluation Environment, are evolving simultaneously with the DSME evolution, which makes it difficult to precisely define the logical context within which the DSME will exist.
2. The breadth and scope of distributed systems analysis precludes any attempt to completely automate the process.
3. Performance measurement can be self-interfering (i.e., the effect of the measurement distorts the performance being measured).
4. The generalization of systems (also referred to as abstractions) is a difficult issue which cannot be completely solved under the current effort.
5. Complete portability is difficult to achieve, especially given the close coupling of performance monitoring tools to operating systems and hardware architectures. The implementation of the DSME Concept Demonstration system assumes strictly that the host system is a VAX system running the VMS² operating system.

2. VAX and VMS are trademarks of the Digital Equipment Corporation.

The ultimate goal of the DSME concept is to provide a comprehensive tool which addresses each phase of the experimentation process, where the experiment domain is distributed system analysis.

This goal for DSME, while reasonable from the perspective of a required capability for supporting distributed system development, represents a challenge both technically and financially. In order to confine the problem to one that can be effectively addressed while still providing a useful capability, PGSC defined a subset of the overall DSME concept, referred to as the baseline DSME. The baseline DSME focuses on providing a common simulation environment for developing simulations for distributed systems analyses. In particular, the baseline DSME can provide support for:

- Simulation modeling;
- Simulation performance evaluation; and
- Simulation experiment analysis.

A more detailed discussion of the DSME requirements and the baseline DSME objectives is included as Section 3 of this report.

1.4 SUMMARY OF BASELINE DSME DESIGN CONCEPTS

Once the requirements for the overall DSME concept had been established and a manageable subset had been identified as the baseline DSME, concepts for the design of the baseline DSME system were developed. That design is presented in Section 4.1 in the form of data flow diagrams. The system functional design is based on two high-level functions: incorporation of a simulation under study (SUS)³ into the DSME environment, and execution of experiments using the SUS.

A number of key technical issues were addressed as part of the design process. Some of these issues evolved directly from attempts to deal with the constraints identified in the preceding section. These issues include:

- Identification of the DSME context;
- Identification of the DSME analysis capability context;
- Identification of the potential for DSME automated analysis;

3. In discussions related to the overall DSME concept, SUS indicates the system under study; in discussions related to the baseline DSME, SUS indicates the simulation under study.

- Evaluation of performance measurement self-interference;
- Determination of the potential for simulation generalization;
- Determination of the validity of experiment stimulation;
- Use of object orientation;
- Definition of DSME scope;
- Software configurations; and
- Data collection approaches.

The data flow representation of the baseline DSME design, and a more detailed description of the design issues, are included in Section 4.

1.5 OVERVIEW OF THE DSME CONCEPT DEMONSTRATION SYSTEM

As the design for the baseline DSME system evolved, it became apparent that even implementation of the baseline capability exceeded available resources for the effort. Thus the decision was made to implement a system which would demonstrate the key DSME concepts, and provide a capability that could be used as a prototype to allow greater insight into the potential utility of a DSME capability. The key concepts that were demonstrated include:

- A generalized, object-oriented user interface;
- Experiment definition tools; and
- Data collection from executing experiments.

Figure 1-1 shows the architecture of the DSME Concept Demonstration system. The User Interface provides the interface between the user and the DSME database, which stores information about systems and experiments. The Experiment Preparation component prompts the user for experiment information and creates the files necessary for experiment execution. The SUS in the DSME Concept Demonstration System is the Simulation Driver Integration system, which is an air surveillance model which includes two simultaneously executing models of different aspects of the air surveillance situation. The Loader takes data generated by the data collection probes in the SUS and loads the data into a database. The Reporter then creates formatted reports of the data stored in the database. Section 5 contains a more detailed description of the DSME Concept Demonstration system.

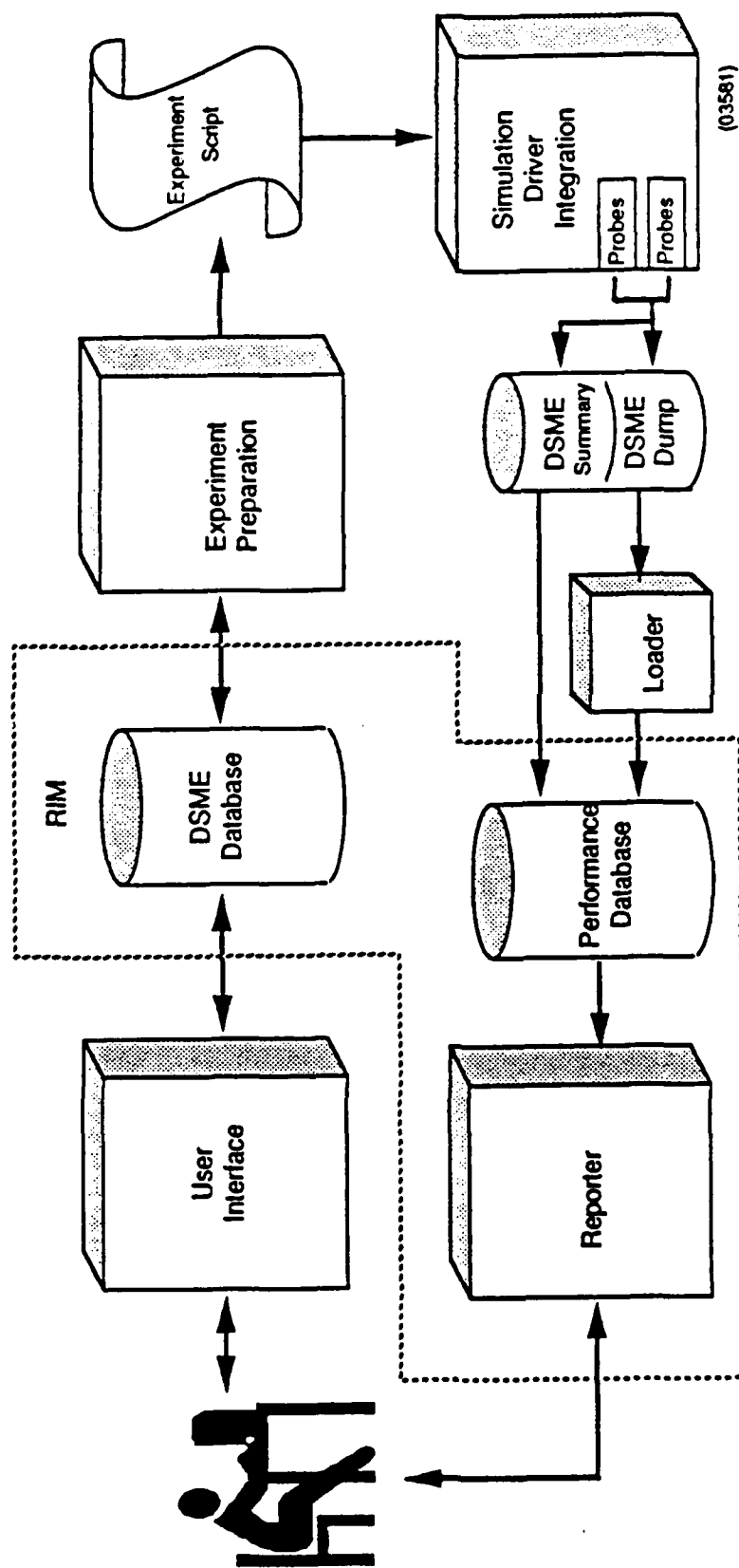


Figure 1-1, DSME Concept Demonstration Architecture

1.6 SUMMARY OF LESSONS LEARNED AND FUTURE DIRECTIONS

As a result of the work performed on the DSME contract, the following observations have been made:

1. The design concepts and the data flow charts are a useful building block for subsequent development.
2. The object-oriented user interface provides a useful baseline for database manipulation.
3. Much of distributed system performance analysis requires the same data to be collected as non-distributed system analysis.
4. The generality of the data collection mechanism remains a difficult issue.
5. Commercial DBMS and statistical packages are an appropriate part of a DSME environment.

The team also identified five steps that could be taken, working from the existing baseline of this report and the Concept Demonstration system. These steps are :

1. Further define the relationship between the DSME and the DISE.
2. Develop instrumentation tools.
3. Insert more powerful relational database management systems and statistics packages.
4. Develop the concept of abstraction to provide greater generality.
5. Conduct a test on another System Under Study.

After these steps are taken, sufficient information would be available to implement the remainder of the baseline DSME concept as well as to continue evolution of the overall DSME concept.

A more detailed discussion of these issues is the topic of Section 6.

2. DSME CONTEXT

This section sets the context for the DSME effort by addressing key background issues. During the development of the DSME context summary, PGSC/Harris and RADC defined an overall context for the proposed DSME and defined the framework of that system. They also established the particular goals of the current effort in relation to the overall target system, which is expected to evolve through a long-term development program. The current effort resulted in the development of the proof-of-concept demonstration capability described in Section 5.

Section 2.1 describes the background (context) of the DSME effort, which provides the initial insight into the requirements that DSME should ultimately fulfill. In order to understand the role that DSME can play within that context, it is useful to define the various processes that support distributed system development; that is the topic of Section 2.2. Finally, Section 2.3 describes the existing capabilities relating to the DSME objectives.

2.1 DSME BACKGROUND

This subsection summarizes the context within which the DSME concept has been formulated and within which the DSME system must effectively operate. This section begins with a discussion of the fundamental motivation for the DSME effort, namely the developing importance of distributed computer systems, and the need for distributed system development and analysis tools. Applicable distributed systems efforts in the RADC environment that address this critical technology area are then described. Finally, the role of the proposed DSME within such a distributed systems development context is discussed. (Much of this section is summarized from the Functional Description,¹ which includes a more detailed analysis of these topics.)

2.1.1 *The Need for Distributed Computer Systems*

The past decade has seen a dramatic rise in the development and application of distributed computer systems. This rise can be attributed to a number of factors:

- Rapidly increasing user demands for compute power;
- Potential physical limitations on individual hardware systems;
- Reliability requirements;
- Expandability requirements;
- Flexibility requirements; and
- Economic considerations.

Each of these factors is addressed in a paragraph below.

1. Brown, C.L., Frantz, F.K., and Wheeler, D.A., *Distributed System Modeling Environment Functional Description*, PAR Government Systems Corporation, PGSC Report 89-64, 2 March 1990.

Significant computer hardware performance improvements have been made in recent years, and increasingly powerful computer systems have become available to a growing user community. These improvements have generally been realized across the entire spectrum of computer system hardware components, including high speed central processing units, larger and faster memory, faster and larger secondary storage devices and controllers, more reliable and greater bandwidth communications between computers, and more intelligent and capable network management of multiple computer systems.

This trend of drastic improvements in hardware performance, rather than merely serving to meet user demand, has led to higher expectations in the software user community and has led to increased demands on computer systems by software developers. These performance requirements are for faster, more reliable, and more secure computer operations. Performance requirements associated with the components of the Strategic Defense Initiative are good examples of performance requirements that are extreme from the viewpoints of both speed and reliability.

Despite the improvements in hardware performance, many existing and expected computer technologies have practical and theoretical limits which are being approached. The speeds of silicon-based processors have upper bounds which are rapidly being reached in state-of-the-art chip development. Furthermore, issues of reliability are not easily addressed in hardware without sacrifice in performance speed. Activities supporting reliability add to the overhead cost of applications processing, thus reducing processing throughput. Also, fault-tolerant processing schemes involve overhead costs, such as the processing and communications times required to conduct a polling/voting protocol within a multiprocessor environment. Finally, fast yet reliable processors remain extremely expensive.

Distributed system capacity may usually be expanded by the addition of processing or peripheral nodes, an option not generally possible with other concurrent systems such as multiprocessors. In such cases, software which does not meet its design specifications may still achieve performance requirements, or a database which has grown beyond expected bounds may be accessed as required.

A distributed system is flexible since differing processing capabilities (such as LISP machines) may be linked with conventional processors, and processing configurations may be altered to adapt to current software or hardware requirements. For example, an application may be executed on four instead of five processors while one processor is undergoing maintenance.

Another factor contributing to distributed system cost-effectiveness is the price-performance ratio (dollars per MIP), which is lowest for smaller processors, such as PCs. This makes collections of smaller processors a more attractive alternative to powerful mainframes. However, this advantage may be deceiving, since performance losses in system operations and the costs of developing distributed applications may offset the price-performance advantages.

2.1.2 *Approaches to Achieving Distribution*

The basic method used to attain continued system performance improvements despite hardware limitations involves the distribution of a processing task among several processing elements in a computer system. If a task can be segmented and different components executed simultaneously, the speed limitations of a single processor can be avoided, and speed and reliability improvements may still be achieved. This approach is called concurrent or parallel processing and may be implemented in a number of ways, including vector (pipelined), array, multiprocessor, and distributed processing.

This trend toward the implementation of parallel processing depends on some software considerations which are currently being addressed in the development of concurrent processing systems. A number of factors determine the effectiveness of concurrent computer systems, including the:

- potential for the task to be concurrently processed,
- performance of the concurrent processing components, and
- performance of the communications between the processing components.

A distributed computer system architecture is one approach which is frequently employed to achieve the performance benefits of parallel computation, and is one of the focuses of the DSME effort. Distributed systems provide distinct overall advantages for many computer system implementations, including high levels of reliability, flexibility, and cost-effectiveness; the details of these advantages are discussed in this subsection.

The term "distributed system" is not well defined, and is often applied to systems as diverse as uncoordinated computers in a loose network, and coordinating computers operating under a distributed operating system. For the purposes of DSME, a distributed system will be defined as any networked system of computers operating under a distributed operating system (DOS) or operating under the control of a distributed application. This extension to distributed applications permits the inclusion of the large number of existing systems which are distributed only through application software protocols. The status of certain other tightly-coupled systems, such as the generic N-cube or Hypercube, was not specifically addressed under DSME.

Distributed processing systems may be characterized by the degree of control which is exercised by the system on individual nodes, classified as loose or tight coupling. Even in tightly-coupled distributed systems, system control is much less than the control intrinsic to such parallel processing systems as pipelined and array processors. Three distributed system categories of interest to this project are:

- homogeneous tightly-coupled configurations,
- heterogeneous tightly-coupled configurations, and
- heterogeneous loosely-coupled configurations.

Although not designed with parallel processing in mind, existing software systems may often contain components in which concurrency may be exploited. Processes, routines, or code segments which might be executed in parallel can be identified through static analysis of system code or run-time performance.

Newer software systems can be designed with the expectation of identifying and exploiting execution concurrency. Utilizing run-time environment features (such as VMS system processes) or programming language features (such as the Modula-2 process capability and Ada tasking), a software designer may specify parallelism in system code.

Future software development systems will incorporate features which automatically exploit concurrency that is inherent in the code. Much like an optimizing compiler alters generated code to improve performance, a 'parallel-optimizing' compiler can identify code segments which may be executed concurrently on a target hardware system to improve performance. Such a system does not require specific designer or programmer attention to concurrency issues, although such attention would generally result in improved parallel performance.

Additionally, significant mathematic and software engineering research is focusing on the development of basic algorithms which are inherently parallel, since most current numerical analysis techniques were developed under the assumption of a serialized processing environment. Such algorithms may be wholly replaced by new approaches which are designed under the assumptions of computational parallelism.

The mechanisms which support the concurrent processing within distributed systems and which exploit concurrency in processing tasks are still undergoing extensive research and development. Many issues of performance in distributed environments are still unresolved and require extensive study and experimentation. For example, it is not always obvious that gains in parallelism will more than offset the overhead processing requirements for task distribution. This is particularly true when communications transmission times are long. The need for basic research is most evident in the area of generalized distributed system applications, since individual applications can directly approach their particular issues such as overhead processing.

Despite the developments in hardware that make concurrent processing attractive, and the developments in software that make it feasible, there is still one element (one missing puzzle piece) required to truly exploit concurrent processing potential. That element is a systematic approach to developing distributed systems, supported by tools. The DSME effort focuses on such tools.

2.1.3 RADC Distributed Systems Efforts

The DSME effort is not being developed in isolation. Nearly all the work sponsored by RADC/COTD is directed at developing distributed system technology. This section presents a brief overview of some of those programs. RADC/COTD is involved in several areas of distributed system technology, including:

1. Distributed operating systems;
2. Distributed databases;
3. Distributed planning; and
4. Distributed system simulation and modeling.

The ongoing efforts in these areas span a range from pure research to the implementation and experimentation with prototype distributed systems. These areas are summarized in the following subsections.

2.1.3.1 Distributed Operating Systems

In terms of distributed systems development, the area of distributed operating systems is probably the one with which RADC has had the longest and most successful involvement. The flagship program for RADC/COTD in this area is the Cronus distributed operating system. Cronus provides an architecture and tools for building and operating distributed applications on a diverse set of machines. Cronus is more accurately identified as a distributed computing environment, since its role as a distributed operating system represents only a partial set of its capabilities. Cronus also functions as an interprocess communications facility, a distributed database system, and a software development environment.

Three aspects of Cronus are of particular interest in relation to DSME. First, applications which run on Cronus represent the type of distributed systems which DSME is envisioned to support. Thus DSME could be used to provide performance analysis support for applications which run on Cronus, or could be used to support the simulation of such applications to determine the implications of distribution.

A second potential link between DSME and Cronus is that DSME could ultimately be implemented as a distributed system application itself. Considering that DSME could be used to collect data from multiple processors within a distributed application, the implementation of DSME in a distributed fashion (co-located on the processors, but with some centralized control to coordinate user input and the overall data collection and analysis process) could be most effective.

The third element of Cronus, relevant to DSME, is its role as a software development environment. Cronus provides the following tools:

- Object-oriented programming paradigm
- Type hierarchy and inheritance
- Specification driven development
- Automated development support

- Libraries
- Debugging aids.²

Since both Cronus and DSME are based on object-oriented paradigms, it will be conceivable at some future point to consider the potential value of interfacing DSME and Cronus. However, since DSME is a long way from a level of implementation that could logically be interfaced with Cronus, this idea is not addressed in any more detail in this report.

Another distributed operating system research project with RADC sponsorship is the Alpha project. Alpha³ is intended to support computers in accomplishing the integration and operation of large, complex, distributed real-time systems such as C³I, battle and combat system management systems, platform management systems, and factory and industrial automation. Key capabilities include real-time operation in a distributed environment, with emphasis on survivability and adaptability.

The potential relationship between Alpha and DSME is virtually identical to that between Cronus and DSME, particularly with respect to DSME providing some type of analytic support to applications implemented under Alpha. However, again it is emphasized that extensive development of DSME is still required before such an undertaking would be meaningful.

2.1.3.2 Distributed Databases

Another major topic of research undertaken by RADC/COTD is that of distributed databases. A number of research projects have addressed various aspects of distributed databases, including security and fault tolerance. These projects, at this point, provide little direct technology support for DSME, since there is no immediate consideration of implementing the databases in DSME as distributed databases. However, as DSME evolves, there is potential for the DSME databases to be distributed (in much the same way that DSME itself could be implemented as a distributed system, as discussed in the preceding subsection).

In addition, the distributed database projects being sponsored by RADC could also be considered as distributed systems whose development could be supported by the DSME environment.

2. Vinter, Stephen, "The Evolution of Cronus," RADC/COTD Technical Exchange Meeting, January 1989.

3. Jensen, E. D., Northcutt, J. D., Clark, R. K., Shipman, S. E., Maynard, D. P., and Lindsay, D. C., *The Alpha Operating System: An Overview*, Archons Project Technical Report #88121, Department of Computer Science, Carnegie-Mellon University, December 1988.

2.1.3.3 Distributed Planning

Distributed planning is an application area in which RADC has recently been significantly involved. RADC/COTD is currently sponsoring an effort to develop a testbed facility oriented toward a real-world problem: generation of the SIOP. This program is the Survivable Adaptable Planning Environment (SAPE). The emphasis in SAPE is to provide a means for maintaining the capability for multiple nodes to participate in the planning process in an extremely hostile environment. Key issues include system-level issues, adaptive planning issues, distributed processing issues, and communications issues.

From the DSME perspective, the primary interest of SAPE is as a distributed system application. It is the best real-world application available at RADC. Here again, DSME could ultimately be used to support this type of application.

2.1.3.4 Distributed System Modeling and Simulation

Since the theoretical bases for distributed system operations are often not analytical in nature, the importance of simulation modeling in such research is obvious. Simulation has proved valuable in investigating the effectiveness of distributed system algorithms, such as process scheduling or system reconfiguration. Simulation is also valuable in the investigation of overall system performance, such as the ability of a system to process a given distributed application in real time, or the actions of a system undergoing reconfiguration after a node failure. Simulation is also valuable in the investigation of system component performance, such as the throughput requirements for given distributed system nodes, or the bandwidth required of the communication network.

Most importantly, simulation permits these types of studies without requiring a prototype or some other implementation of the system. Systems or system concepts that are in formative stages of development can be modeled, and their performance can be predicted. Systems containing thousands of nodes can also be modeled without the expense of implementation. System parameters, such as speed, reliability, and load, can be easily altered to accomplish sensitivity analyses without the need of hardware or operational software modifications.

Despite these capabilities, there are limitations to distributed system simulation. The extreme complexity of distributed systems and the potential interactions of their many components encumber the development of meaningful simulation models.

Recent COTD efforts in the simulation modeling area have included Simulation Driver Integration (SIM DRIVER), Distributed System Simulator 2.0 (DSS), and Internetted Systems Modeling (ISM). SIM DRIVER, developed by PGSC, implemented a distributed simulation capability for command and control applications. DSS 2.0, developed by Harris Corporation, provides simulation capabilities for the performance analysis of local and wide area networks.

The ISM project, developed by Harris, involves the modeling of an SDI battle management distributed computing system, including ground-, air-, and space-based nodes. The simulation system is based upon the existing DSS and includes the extension of that system in addition to the development of the required SDI models.

The experience gained from these efforts and from other RADC simulation efforts indicates the need for improved simulation capabilities. Of particular importance is the need for standardization to assist simulation users and developers.

Many ongoing and projected RADC/COTD projects could benefit either directly or indirectly from the simulation modeling capabilities which are within the domain of the DSME concept. Standardization could improve the utility of existing simulations, decrease the development costs for future simulations, and improve the opportunities for merging existing and future simulations. In particular, simulation tools under such an environment could be used to evaluate the performance characteristics of the many distributed systems and components that are undergoing research and development.

One COTD area of research, the projected Distributed System Evaluation Environment (DISE), might be combined with the DSME as part of a total distributed system development environment. DISE consists of a hardware and software testbed for the investigation of distributed system concepts, and might serve as a testbed for systems or components which have been refined in the DSME simulation environment. Additionally, implementation of a distributed system in the DISE could serve to validate DSME models. In the reverse case, DSME could be used to examine potential DISE configurations before their implementation in hardware. This potential DSME/DISE relationship should be explored in more detail as a subsequent step in the evolution of the DSME concept.

One effort which falls outside the RADC sphere, but is relevant, is Honeywell's Distributed Computing Testbed (DCT) work. There are many parallels between these efforts and objectives of the overall DSME concept, particularly in the experimentation on actual distributed systems to analyze distributed system concepts and implementations. Specifically, the event-action model of experimentation⁴ seems to be a general basis for distributed system monitoring and may serve as the basis for a distributed system monitoring capability such as the one that might be included in the ultimate DSME context.

2.2 DEVELOPMENT PROCESSES SUPPORTED BY DSME

The preceding section included a definition of the context of DSME with respect to the development rationale for distributed systems and the ongoing research into distributed systems. This section addresses a different context within which DSME exists – that is, the context of processes which support developing distributed systems.

4. Heimerdinger, W., and E. Arthurs, "DCT-A Testbed Approach to Distributed Systems Research," *IEEE Proceedings of the International Conference on Data Engineering*, 1984.

The Distributed System Modeling Environment (DSME) will assist the development and analysis of distributed computer systems by providing support for the simulation modeling of these systems and their components. It is therefore important to understand: 1) the procedures which are utilized to develop distributed systems, and 2) the procedures involved in the analysis of distributed systems. The procedures for developing distributed systems are extremely diverse and exist in various states of automation. The procedures for distributed system analysis are in a similar state of development, and are required throughout the distributed system development process. This section examines these functional context areas, beginning with the distributed system development process.

2.2.1 *Distributed System Development Procedures*

In an "ideal" development process, an analytical model of the distributed system is first developed and evaluated. Next, a computer simulation model is developed and used to further explore the system's or its components' characteristics. Next, a testbed or prototype system is developed to test specific concepts or system capabilities. Finally, an operational system is developed and evaluated in the laboratory and in the field. The design and implementation phases of distributed system development involve all or most of the steps shown in Figure 2-1.

Although each of these steps is not always utilized (and then not always in the sequence described above), in combination they provide a general framework which allows generalizations about the development of distributed systems. This framework will be the subject of continuing research. The components of the framework are summarized in the following paragraphs.

The analysis of single-computer systems is often empirical due to the complexity of such systems. This complexity is due to many factors, including the number of processing and peripheral elements, the number of communications paths and mechanisms, the number of software processes being executed, and the number of timing possibilities which exist among these components. The interactions among these many components are extremely difficult to analyze in detail. Highly accurate mathematical analytic modeling of a single-CPU system is usually impractical or impossible.⁵ The additional complexity inherent in distributed systems further increases the difficulty of the analytical modeling process and may prevent its accomplishment.

Despite this difficulty, most computer system designs are based upon some background analysis work, even if this analysis only covers a portion of the system and its concepts. For example, various aspects of graph theory may be applied to the investigation of optimal distributed system communications network topologies. Analytic modeling, normally the first step in the development process, can provide a rigorous definition of a system and its capabilities which may not otherwise be available. Subjects for analysis include scheduling

5. Ferrari, D., Serazzi, G., Zeigner, A., *Analysis and Tuning of Computer Systems*, Prentice Hall, Englewood Cliffs, NJ 1983.

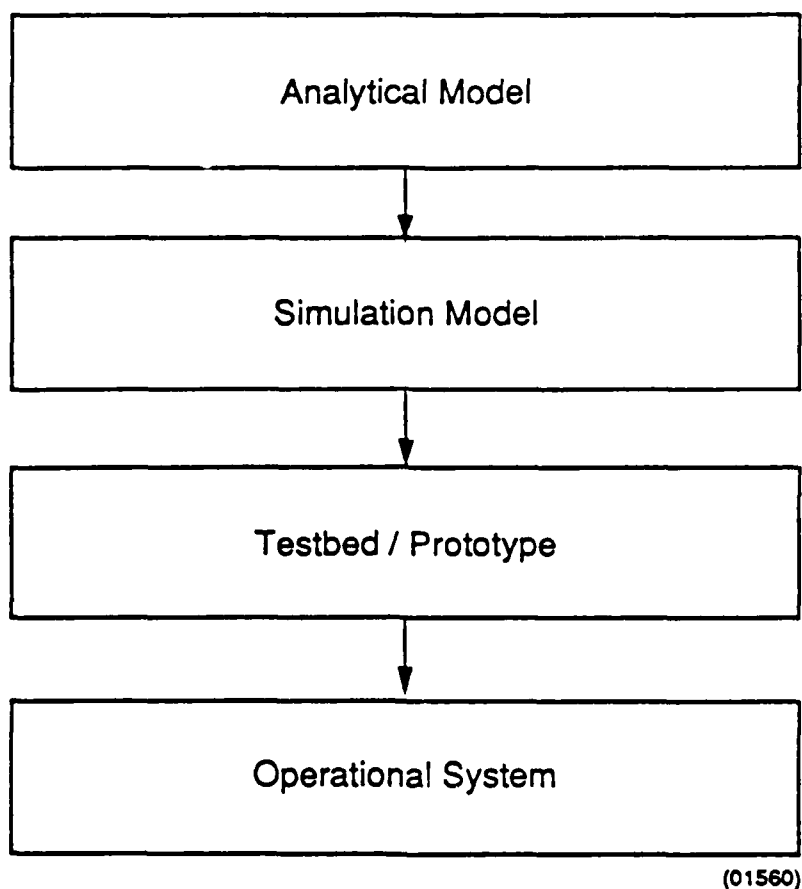


Figure 2-1, Distributed Systems Development Process

and deadlock algorithm development, distributed system topology and message routing optimizations, and reconfiguration algorithm requirements.

Computer simulation, the next step in the development process, is a valuable tool which is extensively used in the study of computer system performance. Simulation models of a distributed computer system can demonstrate the performance of the system under controlled conditions and under conditions which are not attainable (or not measurable) in hardware-based systems. It is natural that the complexity which prevents rigorous analytic modeling also serves to limit the accuracy of computer modeling, yet the technique still has significant value in determining particular aspects of system performance, such as individual component performance or average overall system performance.

Simulation also provides the capability to examine many potential configurations or protocols without investment in less flexible, more costly hardware systems. This is particularly important in an area as new as distributed systems, since many concepts have not reached the implementation stage. This is also important due to the number of potential configurations for a given set of distributed system components.

As the next step of the development process, implementation and study of system prototypes or hardware testbeds may be appropriate to identify or resolve system performance issues. These interim implementations permit the observation of actual system or component performance under realistic conditions, such as software loading or hardware failure. Although in certain cases hardware implementations provide more realistic testing conditions than simulation does, they will also require some sacrifices. Examples of implementation limitations may include the requirement to approximate system loading (where actual software cannot be run on the prototype) and the inability to implement the entire proposed system (where hundreds of nodes may be involved).

The final goal of distributed system development is the fielding of an operational system. Observation of the system testing and operation at and beyond the final stage of development may indicate the need for further modifications of hardware or software components to achieve desired performance goals. The results of the evaluation may also indicate the need to return to the preceding development stages for additional study.

2.2.2 Distributed System Performance Analysis Procedures

Performance analysis is an important and somewhat ill-defined part of the distributed system development process, and is present in some form at each stage of that process. Performance analysis involves the evaluation of system or component performance as the result of analytical modeling, simulation modeling, or system measurements. Performance itself may be evaluated in several categories, with standards frequently established for speed, reliability, availability, and accuracy. The analysis process usually consists of the interpretation of various observations and measurements to establish metrics for a particular performance category or combination of categories.

Measures of performance (MOPs) and measures of effectiveness (MOEs) for distributed systems are not clearly defined in the general case. Various combinations of the above performance categories may be used to establish these measures in particular situations. For example, it is frequently the primary goal of a distributed system to provide specified levels of availability while maintaining a specified system throughput, or processing speed.

In general, analysis of any computer system will involve the investigation of: 1) the system's resources, 2) the control mechanisms for these resources, and 3) the processes which demand the resources. This is also true for the computing of peripheral nodes in a distributed system. System-level performance may be addressed using system-level abstractions of components, summarizing their general characteristics. In other cases, the analysis may need to include the details of component performance and their interaction with other system components, such as the performance of a particular host computer executing a distributed system task.

2.2.3 An Example of Simulation System Performance Analysis

SIM DRIVER can also provide an example for another area of interest – simulation system performance analysis. (This is the role of SIM DRIVER in the DSME prototype demonstration.) As the complexity of simulation support systems and their models increases, execution efficiency becomes an important issue since execution times are often excessive. To address the issue of execution efficiency it is necessary to evaluate the simulation execution process and determine if and where improvements are needed. Although this can be accomplished using simulation techniques, as described above, it is often more effective to actually monitor the execution of a simulation.

The importance of this concept of performance monitoring is evidenced by its incorporation within many simulation support systems. For example, the DGTS simulation support system which is the basis for SIM DRIVER provides a Performance Monitor capability which collects count and timing information of interest to the simulation developer. Collected data includes information on the DGTS system itself, such as the average time required to schedule an event, and on the SIM DRIVER models, such as the average time required to execute each event (such as Move_vehicles) and the number of times each event was executed. Evaluation of this data determines the value of improving the performance of the associated code, system, or model, in order to improve simulation execution times.

The procedures utilized for system performance analysis experiments are similar to those of simulation modeling, and both will normally be conducted concurrently. The performance analysis is often dependent on the modeling techniques and parameters employed in a particular experiment.

After the experiment is completed, the collected data is reviewed by an analyst to determine the answers to the experiment's questions. This in turn determines the direction of further experimentation. Such analysis would start with data manipulation, including standard data presentation techniques like graphics, statistics, and tabular report generation. Evaluation of these reports is a complex process which is currently not automated.

Review of SIM DRIVER simulation and performance data could determine that the synchronization interval should be lengthened to permit more concurrency. It might also show that the TASRAN process should be further divided or otherwise improved to improve the run-time performance of that simulation component.

2.2.4 The Experimentation Process

As a prelude to determining the requirements of a general simulation support system, this section describes the experimentation process itself – not only in terms of the basic functionality to be achieved, but also in terms of the generalized process by which performance analysis is conducted. A general definition of this process and inclusion of that definition in the design methodology, provides the basis for the DSME concept of operations. This approach enables the DSME system to most appropriately and effectively satisfy the requirements of potential users. Various aspects of this generalized process are evident in the example previously presented.

The phases of experimentation presented in this section (see Figure 2-2) are considered applicable for simulation analysis efforts and will be addressed within the environment provided by the DSME tools. Certain phases will not be completely implemented as automated functions, but must nonetheless be considered and evaluated within the DSME tool context. The general phases of an experiment are defined in the following paragraphs

The experiment definition phase involves the (mostly off-line, or manual) processes of determining the questions to be answered by the experiment, determining the model and data collection requirements, and identifying tools for data collection and subsequent analysis.

The experiment implementation phase involves the actual configuration of the experiment, and includes such processes as implementing the collection requirements, specifying tool parameters, and defining experiment control parameters. This process will probably be more automated, and is basically the implementation of the results of the experiment definition process.

The experiment execution phase involves the execution of the software or computer system under investigation with collection or monitoring mechanisms in place. Experiment execution may involve continuous monitoring of the experiment and planned or spontaneous stimulation of the system under study.

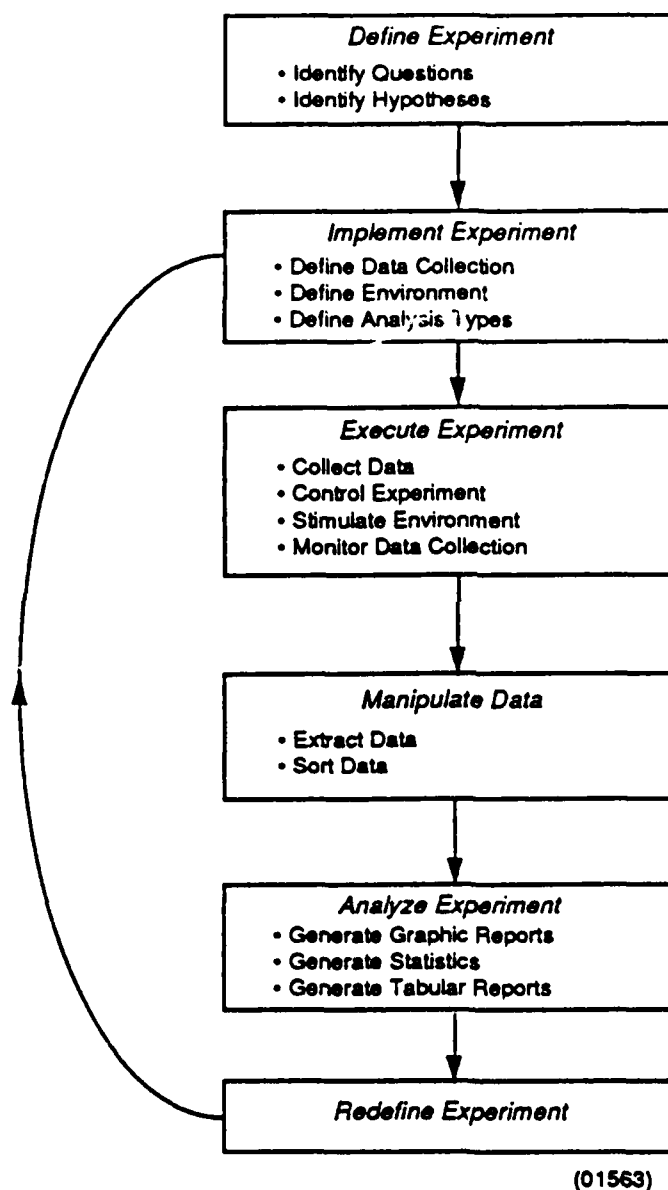


Figure 2-2, Distributed System Monitoring Experimentation Process

Another aspect of computer system evaluation for operational systems is the stimulation of the system to support specific analyses. Actions to vary system loads and inject software or hardware faults can assist the investigation of specific sensitivity issues. This aspect is particularly important given the overall complexity of distributed systems and their analyses.

The experiment data manipulation phase involves the formatting and reduction of data collected during execution, thus enabling a more efficient analysis phase. Such manipulation may be necessary to reduce the bulk of data to be analyzed, thereby permitting more efficient use of processing or storage resources. It may also be useful to focus the data handling on particular areas of interest. This reformatting and indexing of the data may also serve to increase access efficiency. Loading data into a database management system may increase the availability and flexibility of the data and therefore facilitate its analysis.

The experiment analysis phase involves further data manipulation (using statistical, graphical, and tabular approaches) to enable review of the collected data. Reports may be formatted to provide standard tables for review by an analyst. Automated statistical analysis of the data can enhance tabular reports or serve by themselves as summary reports. Graphics techniques provide another view of the data which often serves as a powerful analysis tool.

The experiment redefinition phase involves the modification of the experiment based on the results of the analysis phase. The results may indicate the adequacy of the data collection, the need for additional measurements, the need for a different experimental configuration, or the need for particular experiment stimulation. Many functions of the experimentation phase can be automated in various ways.

2.2.5 The Simulation Process

It is also useful to consider the generalized process by which simulation is conducted. Formulation of a description of this process, and its inclusion in the design methodology, will enhance the relevance of the DSME concept of operations. This approach enables the DSME system to most appropriately and effectively satisfy the requirements of potential simulation users.

This section presents a general process whereby simulation may be conducted. Facets of this process are sometimes ambiguous or are not explicitly defined elsewhere. Therefore, this generalization will serve to standardize the DSME approach.

For the purposes of DSME, a simulation experiment is considered to be a combination of a simulation support system, simulation models (software to be executed by the simulation support system), and simulation data (data to be used by model and system software). Such a combination would be designed and assembled to conduct a particular experiment or group of experiments. The following paragraphs explain the nature of these components, which are graphically displayed in Figure 2-3.

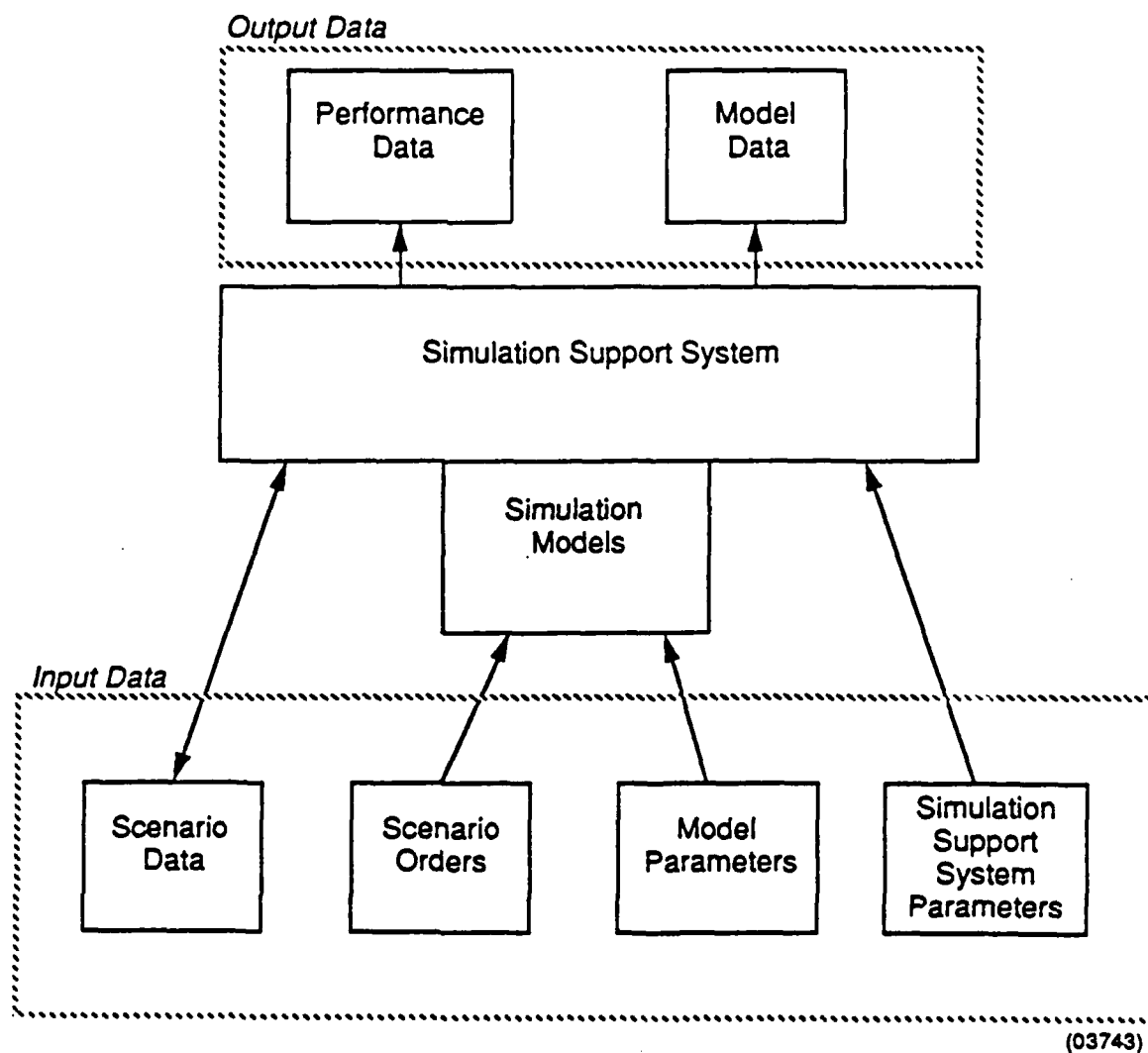


Figure 2-3, Simulation Process

The simulation support system provides an execution environment and basic simulation primitive functions for execution of simulation model software. These primitives include event scheduling, communications, data collection, and database functions. Simulation support systems range in capabilities and complexity from the general SIMSCRIPT and DGTS language environments to the very specific ISM-networked computer system simulation which itself is built in SIMSCRIPT. Advanced features of a simulation support system include simulation data analysis functions and simulation performance analysis tools.

The simulation models are the software routines which accomplish the modeling of desired entities and events. The language used for modeling is dependent on the simulation support system; some systems are specialized for simulation while others are simply variations on conventional programming languages with constructs added for simulation. Some highly specialized simulation support systems include models within their basic structures to support their particular applications. Such models may or may not be accessible to a simulation programmer for modification. Typically a model will be created for certain activities of specific object types, such as ground vehicles, and will be shared by all such object types within the simulation.

Several types of simulation data comprise a simulation system, including: 1) model parameters, 2) scenario data, 3) scenario orders, and 4) simulation support system parameters. These data types are described in the follow paragraphs.

Model parameters are used to control model functionality without altering model code. As an example, the processing speed in the model of a particular type of processor could be provided as a data item rather than being hardcoded or provided as an object parameter.

Scenario data includes the definition of the objects (entities) within a simulation and their individual parameters, such as an identifier, object type, and object capacity. For example, data could be provided to specify a number of processors in a given simulation, and indicate their types and associated memory sizes. The type of processor indicates which model or portion of a model would be used to simulate the processors' activities.

Another type of scenario data, scenario orders, specifies certain actions which will occur at particular times in the simulation. These orders can include initialization actions, such as initial locations of processors in a network, or subsequent actions, such as the failure or destruction of a particular processor at time 'x.'

Simulation support system parameters are used in system code to control run-time characteristics for the simulation. Examples of system parameters which may be data driven are such items as input model data file names, collected output data file names, memory size, and process synchronization intervals.

Not all simulation support systems provide such complete capabilities (by not incorporating, for instance, such concepts as scenario orders), but most may be seen as variations on the general descriptions just discussed. By defining a simulation experiment as a combination of these code and data components, a highly flexible and powerful capability

is achieved. A wide range of experiments may be conducted by simply changing the component elements. For example, three sets of data files defining network topologies could be used with a distributed system model to define three experiments. By providing additional sets of scenario orders data, even more specific experiments could be generated. Changing experiment data can be as simple as changing a file with a text editor.

New models may also be used for enhancements to modeling validity or improvements in execution performance. Changing models within a simulation experiment can be more difficult than changing data, sometimes requiring some form of recompilation. Additionally, a relationship exists between a model and the scenario and parameter data that may necessitate change of these components if a model is replaced or altered. For example, substituting a multiprocessor model for that of a uniprocessor will require significantly differing input data or parameters describing the scenario objects.

Simulation support system parameter changes may be used for experimentation management, such as varying file names and directory locations. Other alterations may be implemented to enhance system performance. For example, a discrete-event simulation advance mechanism may be shifted from a unit time advance to event advance when no external interaction is planned.

After a simulation experiment, or series of experiments, is conducted, the collected data is subjected to analysis. Although some simulation support systems include analysis functions, in this discussion the analysis process is considered to be independent of the simulation process, and is addressed separately.

2.3 EXISTING CAPABILITIES

No software systems exist which satisfy the objectives of the DSME for generalized simulation support of distributed system development. Several distributed system simulation systems exist, none of which is general enough to include existing simulations. Likewise, generalized simulation systems exist, none of which supports existing simulations.

A number of existing capabilities are pertinent to the DSME effort, categorized in the areas of: 1) distributed system modeling, 2) system performance evaluation, and 3) data manipulation and analysis. A study of existing tools was conducted during Task II, Survey of Available Tools, which investigated the possibilities for incorporating existing tools or their concepts in the DSME.

2.3.1 *Distributed System Modeling*

As previously mentioned, a number of candidates exist for inclusion as DSME subscriber simulations. Examples of potential system-level simulations include the Distributed System Simulator (DSS) and its improved versions, in the form of DSS 2.0 and ISM. Due to its close relationship with the current effort, DSS is described in more detail in Appendix B. Examples of potential distributed system components include AISIM, a computer system simulator, and Network II.5, a communications network simulator.

Examples of potential distributed system environments might include DGTS. Although no specific DGTS models exist to support the evaluation of distributed computing systems, many supporting elements are available and development of additional required elements is a proven system capability.

2.3.2 System Performance Evaluation

In the area of system performance evaluation tools, a number of capabilities are currently available to perform certain of the simulation performance evaluation functions of DSME. These include standard operating system support tools, such as the DEC VMS MONITOR utility, and separate tools, such as the DEC System Performance Monitor (SPM).

Using the DEC VMS MONITOR as an example, we can investigate the potential use of an existing performance evaluation tool. This VMS operating system utility accesses a collection of system performance information for immediate display or subsequent utilization. Data elements which can be collected include hardware state information and software process state information, such as CPU idle and process wait state times. This data can be presented immediately in graphical or tabular displays, or it can be stored in binary form on disk. The DSME could and should utilize such an existing capability for data collection on VAX/VMS systems. DSME could translate its generalized data collection specifications to MONITOR commands to actually collect system data during an experiment. After the data is collected in the known MONITOR format, DSME will be able to retrieve the desired data using its own DBMS, and then reformat it as necessary for subsequent analysis. Such a procedure should be followed to avoid duplication of software development (MONITOR already exists) and to utilize the most efficient data collection process available (it is unlikely that non-DEC programmers could implement VMS data collection nearly as efficiently).

2.3.3 Data Manipulation and Analysis

The category of data manipulation and analysis support includes tools that support the user interface, data manipulation, statistical analysis, and graphic presentation. Again, a number of software tools could be utilized within the DSME, primarily since the analysis functions are so general in nature. Most of the processing involving graphical display of analysis data is not specific to DSME, and can easily be accomplished using a standard graphics package with a DSME front-end process. The front-end translates DSME-specific requirements into package calls for the graphics package. Not only does this avoid duplicate and substandard software development, but it also enhances the expandability of the system through the inherent modularity of the approach. The graphics package could be replaced in the future to provide additional capabilities and would only require changes in the DSME graphics interface.

It is also important to note that standard analysis packages and their interfaces may be applicable to more than just the DSME system in the distributed system development context. For example, the analysis required following a simulation run will be very similar to the analysis required following a monitored execution of an actual distributed system.

The following types of applicable software packages are included in the analysis support area:

- user interface management,
- DBMS,
- statistics,
- graphics, and
- report generation.

Screen management packages are a basic type of user interface manager and provide general tools for data entry, menu selection, and interface design. Some existing tools include the commercial DEC Forms Management System (FMS) and Precision Visual ENTER/ACT, and the Harris Corporation Human Interface System (HIS). As another example, database management systems also provide general capabilities. Many fully capable DBMSs exist, including public domain or commercial versions of RIM and Ingres, and the commercial RdB and ORACLE.

3. DSME REQUIREMENTS

This section defines the requirements for the overall DSME concept, then addresses the issue of scaling the design to a manageable subset of the overall concept.

Sections 3.1, 3.2, and 3.3 discuss the requirements for the overall DSME concept. Section 3.1 addresses general requirements. Specific requirements for the tools that are expected to be part of the DSME are described in Section 3.2. Section 3.3 defines a set of functional requirements.

Section 3.4 discusses a number of factors which constrain the development of the full DSME concept. These constraints led to definition of a subset of the DSME concept, specifically one that deals with support for simulations of distributed systems. This subset, known as the baseline DSME, is identified and described in Section 3.5.

3.1 GENERAL REQUIREMENTS OF DSME

This section addresses in detail the general requirements (sometimes known as the “-ilities”) of the DSME. Many of the general requirements are interrelated, and considerable overlap exists in the interpretation of their meanings. Descriptions of requirements may include or otherwise overlap discussions of other general requirements.

The general requirements are so identified because they tend to apply to a wide range of applications. This discussion defines the relationship between these general requirements and the specifics of DSME. The general requirements include the following:

- Generality;
- Flexibility;
- Modularity;
- Usability; and
- Validity and Testability.

Each of these requirements is addressed in a subsection below.

3.1.1 *Generality*

The DSME system design and its utilization must be general to support its overall goal of permitting its use with different simulation systems. This is the primary challenge of the DSME effort.

Generality can be achieved in a variety of ways, although it can seldom be a complete process. The basic approach is based on data generalization and abstraction which are very similar to object-oriented design. Objects and object classes are identified to provide a structure which accommodates as many expected model and simulation system objects as possible. Likewise, generic events or actions are defined to accommodate those found in models and simulation systems.

The use of generalized collection operations and commands, which will be referenced as archive primitives, provide a consistent analyst or programmer interface for data collection. Actions such as including a timetag with collected data can be integral parts of the primitive collection operation. Parameters associated with the commands can serve to further define the actual function of the primitive operations, and may permit some specialization for specific simulations.

Given such primitives and simulation interfaces to the DSME, a programmer is able to collect data using a statement such as:

Archive_object ('processor', processor_id, all_fields);

in all systems, rather than statements which are dependent on the individual systems. This will simplify the implementation of data collection.

The user interface required for such primitive functions includes a mapping for the named objects, such as processor, and the formats of their data fields, including identifiers and other parameters. Such a mapping itself may be data driven, and supported by a global relational database. Several separate areas must be addressed in the logical design of such a database:

- *System Logical database* – the description of the DSME system database structure, which supports system tools and describes simulation experiment database collection and storage;
- *Simulation Logical databases* – the description of the test database to be implemented for the simulation analysis experiments; and
- *Experiment Extension schema* – a description of a general database logical structures which may be used to extend the utility of the DSME.

Another approach which will support generality within the DSME system is the use of standard commercial or public-domain packages for generic functions, such as database management, graphics, and statistics. By utilizing existing general-purpose packages and avoiding specialized implementation of similar DSME functions, a more complete and more general capability is maintained. This is true since: 1) such packages generally contain more complete functionality than could be expected from a contract implementation, and 2) such

packages are designed without a specific application as a target, and are therefore less vulnerable to changes in the target (DSME) system. Although it is obvious that there is a sacrifice in terms of performance (a function tailored to a specific application can almost always take advantage of certain characteristics to improve efficiency), the restrictions in terms of changeability or expandability that accompany such performance improvements are considerable. Some examples of general package utilization are provided in the following paragraphs.

A database management system (DBMS) should be used to generalize the manipulation, storage, and access of the performance data collected by the DSME. Through the DBMS, the performance data would be stored in a common area using standard relations that could be easily defined and that could themselves be stored within the database. These relations greatly simplify the access of particular data by specifying the criteria for retrieval, and letting the DBMS perform the necessary actions to return the appropriate data elements. All manipulation, storage, and access of the data would be performed in the same way, thus making the interface with the data both general and consistent.

A graphics package should be used to generalize the creation of displays of the performance data, such as bar charts and line graphs. The graphics package provides a standard set of subroutine calls that perform graphics functions. The subroutine calls remain the same on all display systems so that a DSME programmer would not need to change function calls when a different display system is being used.

A statistics package should be used to generalize the statistics functions executed on the performance data for analysis. With a statistics package most, if not all, of the functions needed would already be available. Consequently, as statistics functions are required for DSME analysis, a DSME programmer need not write the routines to perform the function. The programmer need only issue a call to the appropriate subroutine within the statistics package.

3.1.2 *Flexibility*

The rudimentary nature of distributed system research and analysis demands that any associated support system be flexible to support modification and expansion as the field evolves. Flexibility is also essential in system development if the DSME is to begin with a subset of the required capabilities and is expected to expand and otherwise evolve in future efforts. These basic issues are addressed in turn.

The first basic requirement for flexibility is based on the evolving nature of distributed system research. The complexity of distributed system analysis prohibits the definition of all the possible experiments to be run using the DSME system. Modifications may be necessary in the future to adjust to new characteristics in the distributed systems under study. New tools may also be needed for particular experiments, and the DSME system must be capable of being expanded to include the new tools.

The second basic requirement for flexibility is based on the evolving nature of the DSME system itself. One of the initial concepts developed under the effort was that DSME is a broad concept that will require evolution over a period of time. Flexibility is essential to permit that evolution of the initial capability into the fully capable objective system. Factors to be considered in the development include the understanding and availability of component tools. Since the tools which are optimal for DSME functions may initially be either unidentified or too costly to include, provision for inclusion of new tools must be provided. This applies to the implementation of tools in the initial effort as well. Incorporation of these tools in the DSME design should be flexible enough to allow their improvement or replacement in future efforts. This is particularly important in view of the difficulty encountered in defining and implementing truly generic tools.

3.1.3 Modularity

Although often listed separately as a desirable or required system characteristic, modularity actually supports the requirements described above. Modularity, which has been described as 'purposeful structuring,'¹ is a basic concept in modern software design. Decomposition of a software system into modules can accomplish several main goals, including those of reliability, modifiability, and comprehensibility.

Modularity involves the organization of functional or software components into a logical structure whereby components are grouped together with other similar components. Since many logical structures may be defined for most systems, the determination of just which structure will be chosen and the evaluation of which features of system components will be considered to determine similarity can normally be accomplished in several valid ways.

Modularity in the software components designed to implement the DSME system is required to support the other general constructs discussed above. The modular structure of the DSME system supports flexibility and expandability by facilitating the inclusion of new functions and the modification of existing tools.

Modularity also supports portability to different operating systems, if this is ever required for the DSME. For each operating system there may be a module containing only the system-specific functions. When the DSME system is to be installed on a new operating system, it would only be necessary to provide the DSME system with the appropriate system-specific module. In this way the DSME developer need not search through a number of files and lines of code to find where the system-specific functions are, change them to suit the particular operating system, and recompile the DSME system.

1. Ross, D., J. Goodenough, and C. Irvine, "Software Engineering Process, Principles, and Goals," *Computer*, May 1975.

3.1.4 Usability

Usability is arguably the most important characteristic of the DSME system, because a user who finds a system difficult or tedious to use will be less inclined to use the system. The DSME system will be developed to contain a consistent and intuitive user interface, and will be designed to have a consistent, acceptably fast response time.

With the DSME system as an interface, the user will be able to conduct an entire experiment without leaving the DSME interface, although the user may in some cases choose to bypass the DSME interface and use the general packages directly. To retain a consistent and user-friendly interface, the system may need to be simplified; however, its usefulness should not be compromised for the sake of simplification.

Also, given that general packages (e.g., graphics, statistics, database) will be extensively used, an additional level of usability is obtained. Although the DSME system should act as an interface to the general packages so the user does not need to learn the specific commands of all the general packages, the power and generality of these packages are available to those users who choose to use them directly, with or without DSME assistance.

The focus of this effort is, therefore, to maximize both the usefulness and the usability of the DSME system. The previous sections described how the usefulness of the system will be provided by its flexibility, generality, and modularity. This section explains how usability will be provided. To maximize usability, the the system must have the following characteristics:

- a standard interface (to simplify use for the average user),
- a flexible interface (to support varying levels of user experience), and
- an effective response time (to avoid user dissatisfaction).

3.1.4.1 Standard Interface

A standard interface is one of the means by which usability may be enhanced and is one of the most desired goals of the DSME system. The standard interface consists of a generic set of commands or procedures for all system operations. This eliminates the requirement for learning a different interface for individual simulation systems. With a standard interface, the user does not require prior knowledge about the data collection details of the distributed system.

If it is to achieve its objectives, the DSME system cannot be based on a particular simulation system or its constructs. Particular modeling mechanisms, functional components, hardware components, and system services available in one simulation system may not be available on another. Therefore, although a completely generic interface is possible in principle, it is not feasible in practice.

Another difficulty in attempting to develop a standard interface involves the methods used to collect the model and system performance data during a DSME experiment. Experiments will be run on a variety of distributed systems, and the constructs of these systems may differ greatly. The difficulty arises in attempting to preserve the generality of the data collection mechanism while attempting to effectively serve widely differing systems. Preservation of the generality of the data collection devices will be made to the greatest extent possible by using generic collecting methods and utilizing simulation system-specific functions only when absolutely necessary.

3.1.4.2 Flexible Interface

Another desired goal of the DSME system is to have a flexible interface to support system usability. A flexible interface provides full support of system capabilities, assistance for the inexperienced user, and direct access for the skilled user.

One possible approach to the user interface is a menu-and-forms driven system, used for command selection and for data and argument entry. Menus organized in a hierarchical structure provide a logical mechanism for the user to execute various DSME commands or functions. While ideal for the novice user, the path tracing required in a full hierarchy can become burdensome to a user familiar with the system. Procedures to facilitate direct access, such as transfer to a distant menu by name or number, can reduce such problems. Menus which are always available (e.g., pull-down or pop-up) can also support experienced and novice users.

3.1.4.3 Effective Response Time

Effective response time is another primary DSME system goal. Response time is that time required to perform a particular user activity using the DSME system. The time required to perform these functions is in many cases dependent on the user's level of experience as well as the efficiency of the DSME software. This means that the time required for an inexperienced user to perform a particular step using the DSME system will be greater than the time required for an experienced user to perform the same function. Appropriate response time is critical for usability because if certain functions of the DSME system are excessively time consuming, the user will be discouraged from utilizing the system.

Current estimates of appropriate times, and their rationales, are presented in the following descriptions. It is emphasized that these estimates are based upon what are considered typical users and average processing experiments. Variations are expected depending on user experience (both with DSME and with distributed systems) and experiment complexity.

Effective response times are considered important for the following phases of the DSME:

- DSME Familiarization Time,
- Experiment Implementation Time,
- Experiment Run Time,
- Data Manipulation Time, and
- Automated Data Analysis Time.

The time for DSME familiarization is that time required by an inexperienced user to become familiar with the operations and use of the DSME. If a new user finds the system difficult to understand and to learn, he or she will not be inclined to use the system in the future. The example given previously of the use of a menu-and-forms driven interface is one way to facilitate the use of the DSME and to reduce the time needed to become familiar with the subsystem. An introductory session with the DSME could require about one day to complete. After the introductory session, the user should have a basic understanding of the system and its operations.

The time for experiment implementation is that time required to perform such actions as the definition of the data collection requirements and the DSME tools necessary for data collection and subsequent analysis. This implies that the user has already defined the experiment but needs to transfer the information concerning the experiment to the DSME. This phase does not include the actual execution of the experiment; it involves the initialization of the DSME in preparation for the running of an experiment. An experienced user should need about one hour to set up the DSME for an experiment.

Experiment run time is different from the other phases of the DSME since it is determined primarily by the complexity and efficiency of the model, which are not totally controllable. Modeling validity may not permit the improvements in model performance required for performance improvement.

One controllable aspect of execution time is the overhead of DSME support tools, which must not significantly contribute to simulation run time. Their use is justified only if their impact on run time is kept to a minimum.

Data manipulation time is that time necessary for a user to process experiment data for analysis. After an experiment is completed, the performance data is manipulated, which may involve such actions as formatting and reduction of the data to provide a more efficient structure for the analysis phase. For a normal experiment, it could take about two hours for a user, regardless of experience level, to run the programs which manipulate the performance data.

Automated data analysis time refers to the time needed to run the report generation routines on the performance data to make it presentable and to give it meaning to the user. This phase also involves the manipulation, presentation, and printing of the collected data in graphical or tabular form for review, but does not include the time for identifying the location of a bottleneck or solving the bottleneck problem. Completion of this phase could take between one and two hours for an experienced user.

3.1.5 *Validity and Testability*

One of the most important features of the overall DSME will be its validity, which will determine its overall value in distributed system development. Validity is an extremely complex characteristic which can not be fully addressed under this effort, since it is determined by the details of the DSME subscriber simulations. The testability of the DSME can currently be addressed from the standpoint of its component tools and the user interface.

The validity of DSME is based upon the validity of: 1) the simulations which it supports, 2) the specialized analysis applications or macros which are developed to evaluate specific distributed systems, and 3) the performance monitoring tools. Applicable analysis applications have not been developed under this effort, and the validity of subscriber simulations is beyond the scope of the effort. The validity of the performance tools is based upon their ability to collect the required performance data without skewing system performance. This is an issue inherent to performance monitoring techniques, and can be addressed in various ways.

The testability of DSME is based upon two main factors: 1) the ability of the environment tools to accurately present the results of the simulation components, and 2) the ability of the environment performance tools to accurately evaluate simulation system performance.

3.2 TOOL DESIGN CHARACTERISTICS

In the context of DSME and its goal of uniformity, some characteristics of ideal performance tools are discussed below. These characteristics form the initial statement of tool requirements that will be expanded and refined as part of the proposed effort. Identified characteristics include:

- consistent interface,
- generic nature,
- ease of collection,
- effective manipulation and analysis capabilities, and
- efficient storage.

3.2.1 *Consistent Interface*

Perhaps the most important issue in simulation tool development is the creation of a single, consistent user interface wherever possible. The learning time required to exploit a different interface for each tool seriously diminishes the utility of the tools, since it discourages the user from using different tools, and imposes a penalty when transferring from one tool to another. In general, this issue has been recognized as an important concept in software design for any software system; its importance is expanded when it is applied to systems which may be tasked to solve a wide range of analysis problems, such as

those being proposed for DSME.

3.2.2 *Generic Nature*

It is important that system performance tools be generic in nature, and not directly related to a single simulation nor to a single type of processor. Such generality is important in the design of the common user interface. For example, if a simulation system event timing is accomplished at the user level by inserting VAX system service calls, or if it is accomplished as a function within the simulation system executive, such a mechanism will vary when applied to differing simulations or processor architectures. A generic system would provide "START TIMER" and "STOP TIMER" functions which are invoked identically on all processing systems.

It is important to note that, at some level, generic performance tool functions must be translated into processor-system-specific or simulation-system-specific calls. The general nature is required only at the analyst level of the evaluation system, and it is a function of the system design to isolate all specific dependencies.

An analogy can be drawn here to the Graphics Kernel System (GKS) approach used for graphic interfaces. The GKS provides a standard set of subroutine calls that perform graphics functions. For each type of display system, there is a package which provides the system-specific instructions for the subroutine. Similarly, at a minimum, the tools should provide standard performance functions, even if specific implementations differ for functions which are inherent to the operating system.

3.2.3 *Ease of Collection*

It is vital that the tools provide as simple a mechanism as possible for the collection of simulation data. If probes are required within simulation software, their insertion and deletion should be accomplished with a minimum of effort and simulation system knowledge. If permanent probes are installed, they should be easily activated or deactivated to support flexible data collection requirements. If new data types are to be collected, detailed knowledge of simulation physical data structures should not be required of the user.

Not only should the collection mechanism be easily placed in the system, but also the collection requirements should be easily specified. Specification most likely is in the form of particular data items, either extracted from the data generated by the system, or performance data such as procedure execution times and memory utilization. Here an analogy can be drawn with the schema concept of databases. Each user of a database has a logical schema which represents his/her view of the organization of the database. The logical view is independent of the physical organization of the database, so that changes to the physical structures do not require changes to the logical schema. Similarly, the specification of data collection should be logical and independent of the physical organization, so that physical organization can change without requiring a change to the data collection specification.

3.2.4 *Effective Data Manipulation and Analysis Capabilities*

The nature and power of the post-processing capabilities are also critical, and should provide the analyst with a natural, user-friendly interface to simulation data. Post-processing capabilities should provide not only access to data, but also general analysis tools.

The interface should permit the general retrieval of simulation data, and not depend on predefined access paths. Provision of a general query language would be extremely useful for ad hoc queries, and support of user-written routines would be useful for repetitious retrievals. Data access routines should provide data as required for input to post-processing statistical analysis tools, including general tools such as SPSS.

The data manipulation tools should include statistical manipulation capabilities and the ability to provide both tabular and graphic presentation of the data. A report generator capability could support user-defined formats for any tabular output. Output data should be transportable, permitting manipulation in other contexts or on other processors, including microcomputer and graphics workstations.

3.2.5 *Efficient Storage*

Although power and functionality are important to the utility tools, their own performance is also an important issue. It is critical that the provided data collection techniques (despite their obvious processing and I/O requirements) not significantly impact simulation performance. It is also important that data storage size be as efficient as possible, since data collection requirements can be extensive. Data storage should also provide accessibility and portability through well-defined, accessible formats.

3.3 FUNCTIONAL REQUIREMENTS

To conduct a complete experiment on a distributed system, the general functions described in the experimentation process are required. Additionally, various environmental control and management functions are needed. Four general DSME functions have been defined to automate the general experiment functions. These functions include:

1. defining the experiment,
2. implementing the experiment,
3. executing the experiment, and
4. manipulating and analyzing the data.

Each of these functions is addressed in a subsection below.

3.3.1 *Experiment Definition*

The definition of the experiment is a strictly manual process in which the researcher determines the objective of the experiment (typically in the form of a hypothesis to be tested, an optimum value for which to conduct a search, or parameters against which to

conduct a sensitivity analysis). Based on this objective, the researcher then defines the test or simulation runs required to conduct the experiment, and the measures of effectiveness and performance that must be collected and analyzed to fulfill the experiment objectives.

Since Experiment Definition is a strictly manual process, there is no DSME functional requirement to support it.

3.3.2 *Experiment Implementation*

After the goals of the simulation experiment (i.e., what will be evaluated during the experiment) have been determined, the next DSME step in preparing for an experiment is experiment implementation, which involves specifying the hardware and software environment on which the experiment will be run, and detailing the data collection that will satisfy the desired goals. The user should also specify the types of output desired. One advantage of specifying the output reports prior to experiment execution may be that selection of an output report that requires more data collection than was requested will cause DSME to flag the discrepancy and request corrective action. Another reason for specifying the output reports beforehand may be to further tailor the data collection to omit any unnecessary collection of data.

In the experiment implementation phase, the following automated functions are expected:

- Configuration of Hardware/Software,
- Specification of Data Collection,
- Specification of Output Reports, and
- Specification of Experiment Control.

3.3.3 *Experiment Execution*

Once the specifications have been completed, the next step is to execute the experiment. The user may control the experiment to some extent, by designating the start of the experiment, specifying how often performance data is collected, turning on and off DSME processes, and establishing the termination of the experiment. The user can also stimulate the experiment by changing the distributed system's environment. Such actions as turning off and on system processes, changing the system workload, changing variables in processes, creating software failures, or deleting files during the execution of the experiment will allow evaluation of the impact of each action on the system under study, as well as an evaluation of the system's fault tolerance. The user may also request on-line monitoring of the system in order to view the utilization of particular resources as the experiment progresses.

In the experiment execution phase, the following automated functions are expected:

- Experiment Control,
- On-line Monitor.

The DSME will include collection tools that access and accumulate data during an experiment. These collection tools will extract and store the necessary performance data at required times during the execution of the experiment.

3.3.4 *Data Manipulation and Analysis*

At the completion of the experiment, the performance data may be stored in a format which is optimal for storage efficiency and not optimal for analysis, such as in sequential binary records on disk. The user may now manipulate the data into a form that will be more efficient for analysis. The user does this by defining the types of manipulations to be performed on the performance data. Examples of types of manipulations include transferring the performance data from binary form on disk to a formatted database, extracting only the performance data between certain times, sorting the performance data in a particular order, or reformatting the data into another order. The user can then process the data by executing the manipulations specified. The manipulation may involve formatting, data reduction, sorting, creation of a database, or loading of a database.

After it has been processed, the performance data is prepared for the analysis step. In this step the user may specify the types of output desired from the performance data. The standard output reports or the output reports defined prior to the execution of the experiment may be selected. The user also has the option to define or create a particular format of an output report. After the output reports have been selected, the user may generate the reports. If the generation of the reports is time consuming, it may be possible to run the generation off line.

In the data analysis phase, the following automated functions are expected:

- Data Manipulation
- On-line Data review,
- Statistical analysis,
- Graphical presentation, and
- Report generation.

In the experiment analysis phase, the following automated functions are expected, all based upon the experiment implementation capability:

- Hardware/Software Configuration,
- Data Collection Specification,
- Report Specification, and
- Experiment Control Specification.

The DSME will include component tools which support the performance evaluation of simulation systems. These tools will permit the evaluation of both hardware and software associated with the simulation. In the hardware area, DSME will collect data that reflects the performance of appropriate hardware devices such as processors, storage and

communication devices, and communication media. In the software area, DSME will collect data reflecting the frequency and duration of execution for appropriate model and simulation system routines.

The DSME will include an extensive collection of tools for the manipulation and summarization of data. Such tools support the data collection tools and form the basis for subsequent data analysis tools. The basis will be a general database management system which supports the programming of specific DSME applications.

One type of DSME data manipulation tool will provide the capability to perform statistical functions on the performance data, thereby facilitating analysis of the performance data. A statistics package, containing standard statistics routines, will be incorporated, if possible, into the DSME to provide generalization and standardization.

One type of DSME data manipulation tool will provide the capability to present the performance data in a graphical manner. Examples of graphical forms include scatter diagrams, bar charts, and Kiviat diagrams. The performance data will first be manipulated using the statistical functions mentioned previously and may then be presented in a graphical manner. This form provides a meaningful representation of the performance data to facilitate subsequent human analysis.

One type of DSME data manipulation tool will provide the capability for automatic report generation, another form of presenting collected data. With this tool the performance data is presented in tabular textual format. The statistical functions mentioned previously may be performed on the data and presented alone as a tabular report or presented in conjunction with other information such as routine execution counts, routine execution times, direct I/O counts, or number of page faults. These reports may be displayed or printed for use in analysis.

The DSME will include tools which directly support the analysis of data. The analysis elements, which are expected to extend beyond the capabilities of general data manipulation tools, provide for selection and analysis of the reports described above, and for the generation of system-level measures of performance (MOPs) and measures of effectiveness (MOEs).

3.4 ASSUMPTIONS AND CONSTRAINTS

Various assumptions and constraints must be factors with DSME development, since the time, resources, and potential rewards of a given system are limited in any system development process. This section documents those constraints and their potential impact on DSME evolution.

DSME Context

The DSME is currently under development and its developers are aware that the system could well be part of a more encompassing environment, such as the Distributed Systems Evaluation Environment. Since many of the coordinating elements of such an environment are not yet well-understood, this awareness of a larger context cannot significantly affect the design and implementation of the DSME. Wherever potential interfaces are expected, and where they can be supported without significantly impacting the progress of the current effort, they will be included in the DSME design. To build on this effort, a subsequent step is recommended – a specific study to define the proper relationship and interface between the DSME concept and the DISE (see Section 6.2.1).

DSME Automated Analysis

The concept of distributed system analysis is sufficiently ill-defined and of such a broad scope that it is not possible to include its automation under the current effort. The DSME will certainly require tools that will support the analysis process, such as report generators and graphics, but will initially provide only limited versions of these functions. The only significant additional support provided by the DSME will be a 'macro' facility, which permits the specification of groups of commands or reports in a format that may be re-used or modified by an analyst.

Performance Measurement Self-Interference

Although this is a crucial factor in performance monitoring, it is believed that the role of performance analysis in the DSME is adequately served by a system that limits but does not absolutely minimize this interference. If interference becomes a critical factor, improvements in system implementation may be made in future DSME development efforts.

Simulation Generalization

The concept of simulation generalization is a significant undertaking that cannot and need not be totally achieved. Approaches toward generalization will be started in the DSME effort, and these initial steps will serve as the basis for subsequent tools and further development.

Experiment Stimulation

The concept of experiment stimulation is not considered essential in the initial DSME system, due to the previously mentioned problems with experiment validity. Since the SIM DRIVER simulation system does include features that support stimulation through user interaction, some initial steps toward providing a generalized capability will be designed and implemented.

Software-Based System Assumption

To provide the generality and portability required of the DSME performance evaluation tools, it is assumed that all data collection will be accomplished in software rather than hardware. Utilization of hardware monitors will restrict the usability of the system and will not sufficiently increase the overall effectiveness.

Generality/Portability Constraints

Due to the complexity and varying development of distributed system simulations it is impossible to provide a DSME with total generality and portability. New concepts and data types will certainly force the alteration of at least some basic system design assumptions sometime in the future. The DSME design effort will therefore attempt to deal with the state of the art and anticipated developments in distributed system simulation, providing as much generality as possible to accommodate new developments.

Existing Software Assumption

One method for supporting generality and portability in DSME is to rely on existing software that already supports these features. For example, database management systems that provide general data capabilities are available. To develop software with such capabilities for DSME is a tremendous and unnecessary task; the resulting software might be more directly applicable to DSME and DISE, but would be restricted in capability and expensive to develop.

It is assumed that maximum appropriate use will be made of existing software packages to implement DSME functions, as determined by package capabilities and costs. During Task II, packages to be included were identified, and are expected to include the following functions:

- DBMS,
- statistics,
- graphics,
- system performance data collection, and
- user interface management.

In addition to these constraints, it is worth noting some observations which do not particularly constrain the design, but do effectively shape and characterize the design.

Object Orientation

After initial review of the proposed DSME concept, it appears that the original PGSC/Harris approach to simulation generalization included many features of object-oriented design. The general simulation constructs of entities and events are analogous to the objects and methods of object-oriented approaches. Continuation of this approach should enable DSME to remain applicable to continuing efforts in distributed system design and development.

DSME Implementation Scope

The initial scope for the DSME, as defined in Section 2, focuses on the SIM DRIVER test case, with additional interest in the DSS as a subsequent test case for future efforts. This limits the current scope, but DSME design decisions will require the inclusion of the impacts of potential future applications. Investigation of the applicability of the DSME concept to SIMSCRIPT simulations must be continued.

Resource constraints have made additional limitations in the DSME implementation scope necessary, including the limitation of the inclusion of general-purpose tools such as statistics generation packages, and the inclusion of the entire Sim Driver system within the DSME. It has been judged inappropriate to attempt to generate significant amounts of general-purpose software under the DSME, since these most often already exist in commercial or public-domain packages. It is more reasonable to incorporate existing packages rather than attempt to duplicate their functionality. This allows more effort to be expended on new DSME features and function. Limiting the SIM DRIVER test case to the DGTS components is recommended due to the lack of relevance of the FORTRAN-based TASRAN simulation to future distributed system developments, and the relatively high cost of incorporating the additional simulation.

3.5 BASELINE DSME OBJECTIVES

The discussions thus far have taken a very broad perspective of what the DSME could be and its requirements for addressing the entire spectrum of distributed systems development support. However, as previously noted, this concept exists on a far grander scale than can be addressed under this effort. In fact, the potential scope of DSME was one of the most difficult issues to address under this effort.

For this reason, attention is now focused on some specific aspects of the DSME concept which can generally be supported by existing technology. This set of capabilities is referred to as the baseline DSME. The baseline DSME concept is discussed first, then the overall objectives of the baseline DSME (in terms of simulation and distributed system analysis) are defined, and finally, the more specific objectives of this specific contract are defined.

3.5.1 The Baseline DSME Concept

The constraints defined in Section 3.4 indicate that the overall scope of the DSME concept, described thus far, is well beyond the scope of what could be accomplished under the current effort. In fact, even a comprehensive design for such a wide-ranging capability is beyond the resources of the current effort. It is necessary, therefore, to focus on a specific subset of the DSME concept that can be developed in more detail, in order to have some concrete product at the end of the effort. Thus, the study focuses on that aspect of DSME which supports the simulation of systems. This subset is identified as the baseline DSME capability.

This baseline was selected because it fit most closely with the target system that was to be used as the test case – the SIM DRIVER simulation system. In addition, support for simulation occurs earliest in the distributed system development process, so that these tools could be used most effectively in the early stages of distributed system development, where more cost-effective design decisions can be made. For the remainder of this report, the discussion focuses specifically on the baseline DSME, unless otherwise stated.

It is evident from the study of simulations in general and of simulations related to distributed systems in particular, that many common elements exist among the various simulation systems. The baseline DSME is envisioned as an environment that supports the simulation of distributed systems, exploiting the commonality of simulations and providing a coherent set of data collection and analysis tools. This set of tools, consolidated into a cohesive, consistent user environment, will provide support for simulation developers and users. Although such a system will have general applicability to simulation modeling, it will specifically support the simulation of distributed systems. By providing a common simulation environment, the DSME will address two main issues: 1) the usability of simulation capabilities, and 2) the development of simulation capabilities.

The most obvious improvement provided by the DSME will be the standardization of the interfaces and tools for the simulation user. All simulations adopted or developed by RADC/COTD for distributed systems research will be able to share a common user interface. Additionally, many tools will be common for all subscriber simulations, thus reducing differences in functionality or capability which would otherwise exist. This can significantly reduce learning and operational time, and can also reduce user frustration. Also, significant cost benefits are obtained as a result of improved throughput and, perhaps more significantly, through the use of simulations which might otherwise be ignored due to difficulties or delays associated with their use.

The development of simulation systems is another area where distinct advantages are realized by exploiting the potential commonality of simulation tools. The availability of standard tools for the collection and processing of simulation-related data eliminates the need to include such tools in the development of new simulation systems. Rather, system developers can incorporate the DSME concept in their design, thereby eliminating the need to design and implement such tools as report and statistics generators.

This commonality can also be viewed and used in terms of standard formats for data and messages. The DSME will consist of such standard procedures and data formats as well as software tools. By establishing these standards, the DSME will provide a complete environment for simulation system development. This standardized approach will simplify the exchange of data between distributed system simulations. It will also simplify the overall performance analysis process, as the results of various simulations of system components are used to provide data for more abstract system simulation.

An overview of the DSME concept is presented in Figure 3-1.

Three general types of simulations are expected to be applicable to the DSME, categorized according to the modeling scopes for which they are designed. These modeling scopes are:

- distributed systems,
- distributed system components, and
- distributed system environments.

These types of DSME simulations are presented in Figure 3-2. The reasons for this organization are discussed in the following paragraphs.

The first and most obvious need is the modeling of complete distributed systems. Such modeling will require some abstraction of each of the basic distributed system components and the surrounding environment, combined with more detailed modeling of the distributed system topology, and its control and loading mechanisms.

Modeling of distributed system components is required to provide information needed for the overall distributed system modeling. For example, a single-node computer system model might be used to generate parameters that describe a single processing node in a distributed system model. Component modeling may also be more complex, dealing with such distributed system elements as communications networks. This modeling is by itself of direct interest in distributed system research, for the investigation of component technologies and capabilities. Again, component modeling also serves to provide parameters for distributed system models. The example of the network model could provide average message transmission times or overall throughput rates for a more generalized distributed system simulation.

Perhaps least obvious in the categorization of these simulation types is the role of the distributed system environment model. Just as distributed system component models provide data which is used in system models, the environment models provide similar data which reflects actual environment factors in system models. For instance, the processing load of a ballistic missile defense (BMD) processing system is related to the number of tracked objects (among many other parameters). Rather than approximating this loading factor in a distributed system model, an environment model could provide this data more realistically.

Another use of the environment model would be to model the distributed system within a simulated environment, thus providing the natural feedback between the performance of the distributed system and the environment. Again using BMD as an example, a space defense simulation which includes a model of a battle management distributed processing system could be employed. This would enable the simulation to address certain issues, such as the ability of the processing system to support the requirements of the BMD Battle Manager in terms of the battlespace environment. It would also permit the system

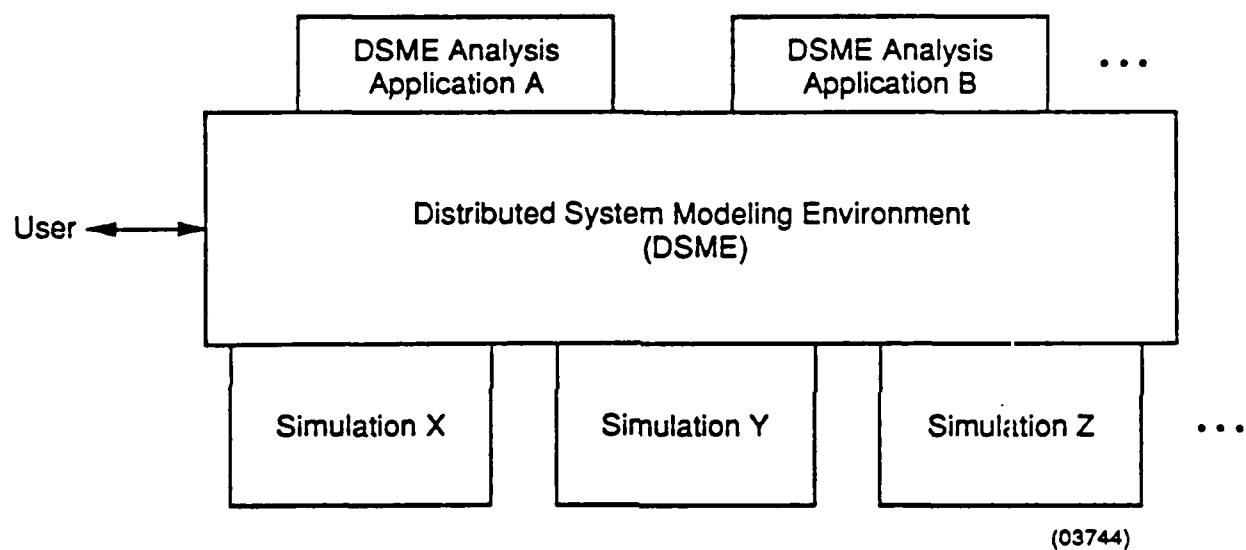


Figure 3-1, DSME Concept

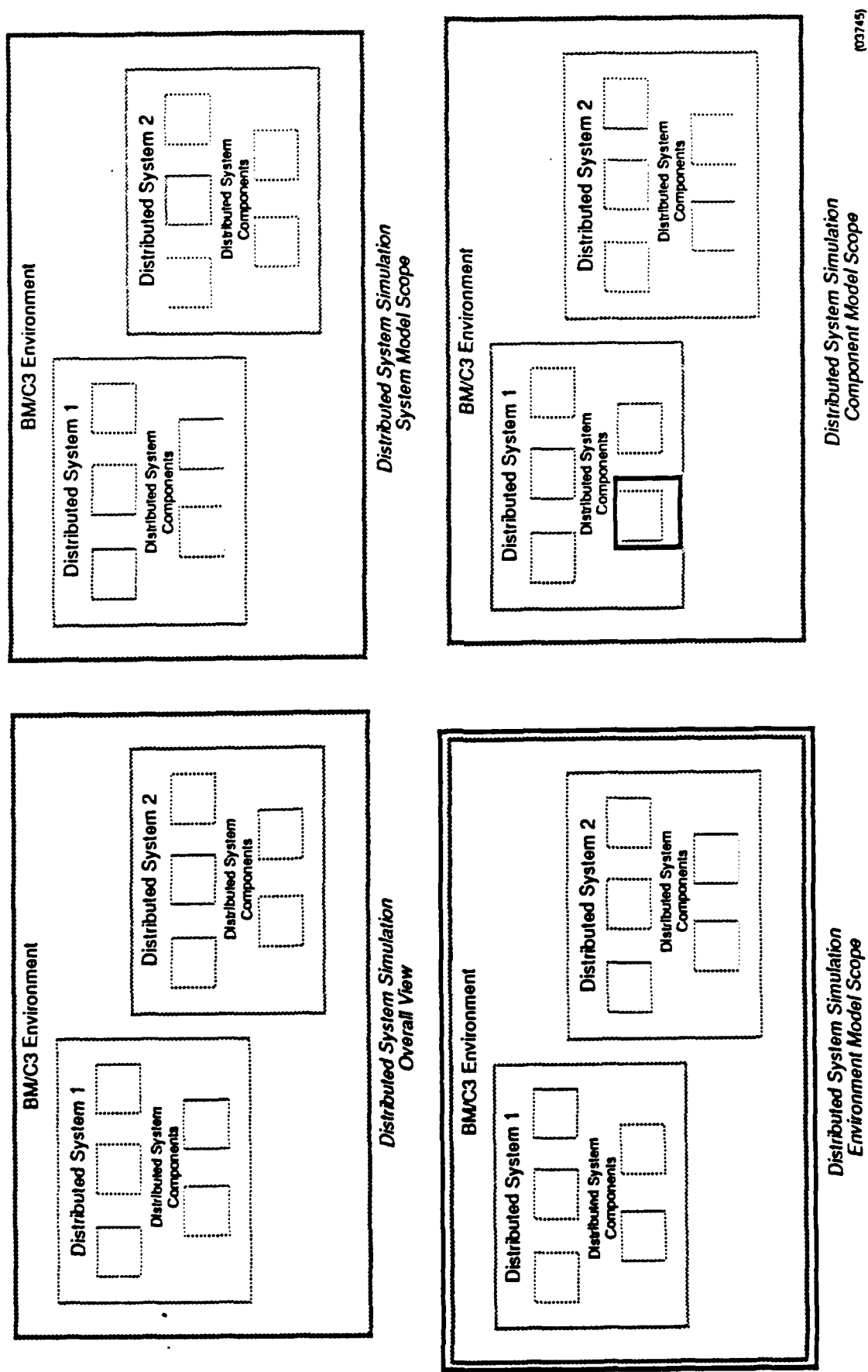


Figure 3-2, Potential Modeling Scopes for DSME

load to vary more realistically, since successful performance of the Battle Manager will theoretically reduce load on the system (e.g., as targets are classified or destroyed).

3.5.2 *Simulation Environment Objectives*

The discussion in this subsection organizes the baseline DSME objectives in the area of generalized simulation support into the following three categories:

- simulation modeling support,
- simulation performance evaluation support, and
- simulation experiment analysis support.

Simulation modeling support primarily involves general support for simulation data collection and data manipulation, and simulation experiment control, and is the base upon which further tools and capabilities are developed. Simulation performance evaluation support involves general support for the performance analysis of simulation support systems and their associated models, again including support for data collection and manipulation. Simulation Experiment Analysis support involves the general support for data analysis, including data manipulation and presentation. These simulation supports are described in the following paragraphs.

3.5.2.1 *Simulation Modeling Support*

Simulation modeling support should consist of a number of automated tools and support software under the basic categories of:

- model development support,
- experiment management support, and
- data management support.

Modeling support provided by the baseline DSME will consist of standard data collection routines and standard data formats for generic and distributed systems-related objects. The standard collection routines and formats will permit simulation developers to build upon existing software rather than requiring them to develop new routines. This not only improves efficiency, but also facilitates the integration of simulations.

Data generalization and standardization are also important approaches for modeling support. For example, in support of modeling efforts the following general data categories may be predefined, and therefore may be specified, collected, and generally manipulated by DSME tools without special implementation:

- **Scenario Summaries:** Tabular reports which summarize certain aspects of a scenario at a specific time.

- **Event Tracking:** Reporting and analysis of data related to simulation events at a specific time, including output such as lists of events currently executing or scheduled, summary counts of event executions, or reports of event timing.
- **Object Tracking:** Reporting and analysis of data describing simulation objects at a specific time, including outputs such as summary reports of object states, counts or states of all objects of a certain type, or counts or summaries of an object's changes of state.

Experiment management support provided by the DSME will consist of coordinated tools to automate the experiment phases of implementation, execution, and data manipulation or analysis. These tools will provide a common user interface for the entire experiment process, and will provide that common interface for the underlying support tools.

Data management support provided by the DSME will consist of a collection of tools which provide capabilities for data manipulation and presentation. Specifically required are capabilities for general database management, tabular report generation, statistics generation, and graphics presentation. Additional tools based upon these capabilities will also be provided in the baseline DSME; such tools would include experiment management data relations and routines, simulation data collection relations and routines, and macro data management relations and routines.

3.5.2.2 Simulation Performance Evaluation Support

The primary goal of simulation system evaluation support is the run-time performance evaluation of support system software and models, which involves the collection of data for executing software and associated hardware. Generalized simulation system evaluation support should consist of a number of automated tools and support software which provide assistance with the following types of evaluations:

- simulation support system performance, and
- model performance.

The capability to observe the efficiency of the simulation support system must be provided within the baseline DSME. This capability is particularly valuable during system development, but is also useful whenever simulation performance becomes an issue of concern. Development of standardized data formats and collection methods will assist this process, although it is not possible to totally generalize them across the many existing simulation systems. In fact, the ability to monitor the internal operations of many commercial simulation systems, such as GPSS, will be limited, and may be determined by access to source code. The simulation tools can address the issue of simulation support system performance by incorporating features which include the support for collection and analysis of such simulation items as:

- event scheduling/execution,
- database access,
- message size, and
- message transmission time.

The capability to observe the efficiency of simulation system models must be provided within the DSME. Development of standardized data formats and collection methods will provide this capability. The performance analyst must determine and insert the probes required for data collection. Again, access to model source code for a given simulation may determine the ability of baseline DSME to support this capability.

Additional simulation performance tools support the analysis of simulation performance as it relates to model data. This analysis is applicable to system performance analysis as well as to experiment evaluation. The following processor analyses may be assisted by baseline DSME tools:

- System loading resource requirements,
- System performance versus scenario activity, and
- Run-time monitor.

Many existing tools satisfy a portion of the functionality required to support these performance evaluation objectives, and may in fact serve as components of a final system. One of the DSME conceptual objectives is to utilize such tools to the maximum extent possible. Various monitoring tools also exist, such as the DEC VMS Monitor Utility which might provide collection support mechanisms within the performance monitoring component of the baseline DSME.

In support of distributed system simulation and analysis, baseline DSME software must provide: 1) data collection mechanisms based in software, 2) experiment definition and control mechanisms, and 3) data manipulation and analysis mechanisms. The data collection mechanisms must be comprehensive, easy to utilize, and of controlled impact on system performance. The experiment definition and control mechanisms must be usable and extensive to permit proper experimentation. The data manipulation and analysis mechanisms must be comprehensive and general.

3.5.2.3 Simulation Experiment Analysis Support

Another objective of simulation is to analyze the experiment data, whether it is model data or system performance analysis data. The analysis support category contains various functions to assist an analyst in the evaluation of collected experiment data. These functions basically involve data manipulation and statistics generation, and include:

- user interface management,
- DBMS,
- statistics,
- graphics, and
- report generation.

Additionally, analysis support includes a function which manages the required application of the previous functions to the collected data. As a simple example, this management function would permit the specification of a collection of output reports to be automatically generated following experiment execution.

It is not the objective of the baseline DSME to support the automated analysis of data to the extent of evaluating experiment MOP/MOE, although this could be addressed in the evolution of the ultimate DSME capability.

3.5.3 Distributed System Evaluation Objectives

The distributed system evaluation objectives of the DSME are limited by the evolving nature of the distributed system analysis process. This is a research area which is in a rudimentary stage of development, and therefore not currently subject to extensive automation. The baseline DSME objectives in this area include the provision of standard data formats, the provision of generalized data manipulation and statistics capabilities to generate reports, and the support for specifying and replicating the reports provided on various sets of data.

The primary capability to be produced in the DSME must support distributed system development. Therefore, another objective of the baseline DSME is the refinement of the general simulation system support to support simulation related specifically to distributed systems. Such an objective is addressed under two main categories: 1) distributed system and component simulations, and 2) distributed system analysis tools.

One of the DSME conceptual objectives is to utilize existing capabilities to the maximum extent possible. In particular, several different simulation systems have been designed to address issues which are important to distributed systems, notably AISIM, and ISM; either might be included as a simulation within the baseline DSME.

4. BASELINE DSME DESIGN CONCEPT

Having established the overall concepts of a DSME, and defined a scope for a baseline capability, we now develop a design to implement that capability. This section is divided into five subsections, each of which provides a different perspective of the baseline DSME design. It begins with a functional decomposition of the baseline DSME that emphasizes the functions and data flows required to provide the capability described in the preceding two sections. Section 4.2 discusses the key architectural elements of the baseline DSME, outlining the various approaches that were considered, and the advantages and disadvantages of each approach. Given the recommended approach, Section 4.3 summarizes the operations (from both the user and system points of view) necessary to fulfill the functions identified in Section 4.1. In Section 4.4, the design is applied to the test case (Simulation Driver Integration) to illustrate the way in which this design would work. This section concludes with a discussion of additional design considerations.

4.1 DESIGN REPRESENTATION

This section documents a functional architecture for the baseline DSME system. A functional architecture is defined as a hierarchical decomposition of the system by functions, which documents functional relationships and interfunction data flows. The purpose of this step in the design is to bound and decompose the problem domain into a structure which can then be used to identify and organize subsequent discussions of key design issues.

This section is divided into three subsections: Section 4.1.1 summarizes the problem to be solved, focusing on the objects which must be manipulated to solve the problem and the actions which provide the basis for those manipulations; Section 4.1.2 presents a series of data-flow diagrams which document the functional decomposition of the baseline DSME system; Section 4.1.3 contains a Data Dictionary which defines the data items identified in Section 4.1.2.

4.1.1 Object-Oriented Problem Definition

This subsection provides a general description of the baseline DSME design which was developed using an object-oriented methodology. The discussion begins with a description of the problem to be solved by the system, and then decomposes that problem into objects and operations on those objects. Each level of decomposition begins with a statement of the problem and a paragraph describing an informal strategy for solving the problem.

In the problem-solving strategy descriptions, the important nouns are expected to be represented as objects in DSME software, and are presented in **bold face**. Important verbs are expected to translate into operations on these objects, and are presented in *italic print*.

The overall baseline DSME problem is summarized in the following sentence. *An environment to support the modeling of distributed systems and their components is to be developed.* The informal strategy for solving this problem at the highest level of abstraction

(highest-level data flow diagrams) is as follows:

A new distributed-system-related simulation is developed specifically for the baseline DSME, or an existing simulation is adapted to operate within the baseline DSME. An analyst utilizes the adapted simulation to conduct modeling experimentation by preparing, executing, and analyzing individual experiments.

The problem of conducting experimentation (second-level data flow diagrams) is summarized in the following sentence. *To conduct an experiment, an analyst is provided generalized support tools for preparing an experiment for execution, executing the experiment to collect or review data, and analyzing the generated data.* The informal strategy for solving this problem is:

To prepare an experiment for execution the analyst must *specify* elements required to execute the simulation (these required elements include model, data, and control inputs, and data outputs). The analyst must also specify the analysis requirements and the DSME control requirements for the experiment (the required analysis requirements include data-collection requirements; DSME control requirements include experiment description, data-collection outputs, and simulation control orders). Experiment specification is accomplished by *creating* an entirely new experiment, or by *replicating* and *modifying* a previously defined experiment. The system *verifies* user input elements where possible, and provides lists of existing elements for user selection where appropriate. The experiment is stored in an experiment library for subsequent *retrieving*.

To execute an experiment, the user *selects* an existing experiment from an experiment library. The descriptions of required simulation execution elements are retrieved from the experiment, and the availability of its components is verified by the system. The user *initiates execution*, and system data collection is invoked from the SUS. Incoming data is *filtered* according to experiment data-collection requirements, *tagged*, *formatted*, and *stored* in a disk file. Selected data may be *routed* to a monitor routine which *displays* or *analyzes* the data. The user *controls* execution, including data collection, through the system software.

To analyze an experiment, the user selects from the experiment library an experiment which has been executed. The descriptions of required execution and data analysis are retrieved from the experiment, and are used to *import* collected data into the DSME DBMS. The system then serves as an interface to the DBMS, which the user accesses by *defining* analysis operations to describe the manipulation and reporting of experiment data.

The functions of the baseline DSME are described here, including quantitative and qualitative descriptions of how these functions will satisfy the requirements of Section 3.

Prior to implementation of a production version of the baseline DSME, the following general characteristics must be considered:

- accuracy and validity,
- timing,
- throughput time,
- response times to queries and data file updates, and
- sequential relationship of system functions.

Accuracy and validity are not primary concerns of the baseline DSME, since the environment consists primarily of support tools. However, one baseline DSME functional area where these characteristics are important is in the role of simulation performance evaluation. As previously discussed, the impact of baseline DSME software on actual system performance affects the validity of any system measurements and analysis. To accurately determine the factors involved in simulation performance, the impact of the baseline DSME software must be either insignificant or effectively factored in performance analysis. In the baseline system under development, neither goal is considered achievable, but each will be a factor in system design.

The only timing concerns related to the baseline DSME are those related to the interference issues discussed in the preceding paragraph. Timing is critical to the baseline DSME only where the precise measurements relating to system performance are involved, and these are beyond the scope of the current effort.

Throughput time is important to the baseline DSME, although there is no single throughput time applicable to the system. Throughput is particularly an issue for database applications.

Response times to queries and data file updates are perhaps the most critical issues relating to the current implementation. Although the performance of the delivered system will focus on functionality and demonstrability rather than on speed, reasonable response times are critical to its eventual utility and acceptance. The general issues of response times for various baseline DSME phases were discussed in Section 3.

Sequential relationship of system functions has been addressed in the baseline DSME design, with emphasis placed on flexibility. Although a certain sequencing (prepare, execute, and analyze) is specified in the overall functional design, the baseline DSME permits concurrent and batch performance of these phases once a set of preparation and execution results exists. That is, once a set of experiments has been prepared, they may be executed at any time, while analysis of previous executions are also being conducted.

4.1.2 Baseline DSME Data-Flow Representation

The following paragraphs contain brief descriptions of the baseline DSME functions as identified in the original functional decomposition. The presentation is in the format of a hierarchy analogous to the decomposition hierarchy. A top-level data-flow diagram of the DSME is provided in Figure 4-1.

Incorporate SUS (1)

Describe and modify SUS to permit its use within the baseline DSME.

A description of the SUS and its components is generated for reference by other baseline DSME functions. Data-collection routines are created to generate output data in a format compatible with other baseline DSME functions, and to control the type and frequency of the collection of this data. Data-collection statements which invoke these routines are inserted within the SUS to initiate the data-collection process at appropriate points during SUS processing. A data-flow diagram of this function is provided in Figure 4-2.

Generate SUS Description (1.1)

Create or modify a description of the SUS object for use by baseline DSME functions.

Generate Data-Collection Routines (1.2)

Not implemented in software, this function involves the preparation of source code routines to be linked with and called from the SUS for data collection. These routines will determine which data is actually stored and will format and store the selected data, serving as the interface between the baseline DSME and the SUS.

Insert Data-Collection Statements (1.3)

Data-collection statements which call appropriate baseline DSME data-collection routines are inserted in the SUS source code. Although not implemented entirely in software, this function will be partially automated through the use of a programmable VMS text editor, which will support templating of collection statements, thereby reducing error probabilities.

Generate Baseline DSME SUS (1.4)

Linking of DSME collection routines and the SUS source code, modified through the insertion of data-collection statements, using a method appropriate for the SUS. Although partially implemented in software, this function will not be fully capable of such activities as trapping errors in the linking process. Rather the function will execute the requisite linkage commands and will require the user to handle other-than-normal termination of the invoked process.

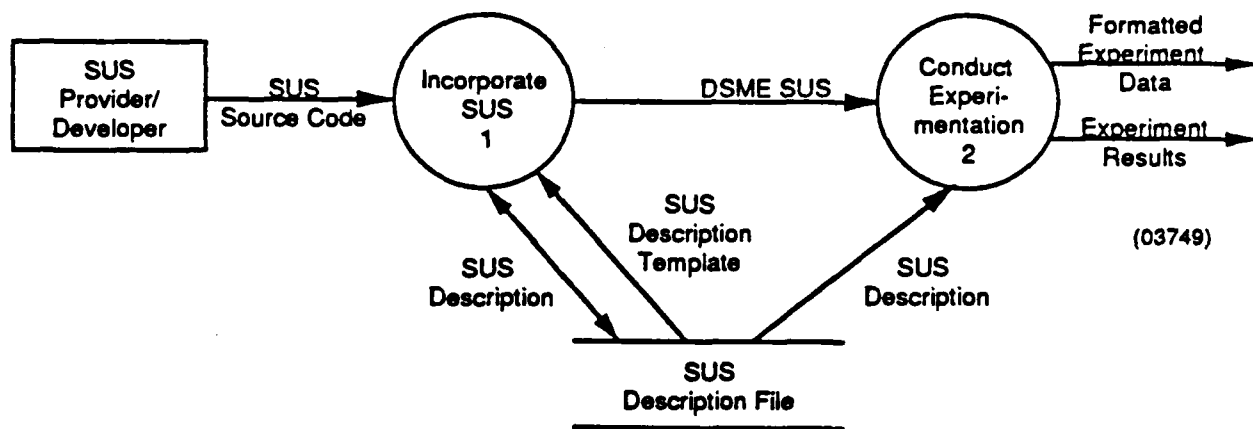


Figure 4-1, DSME DFD - DSME Top Level

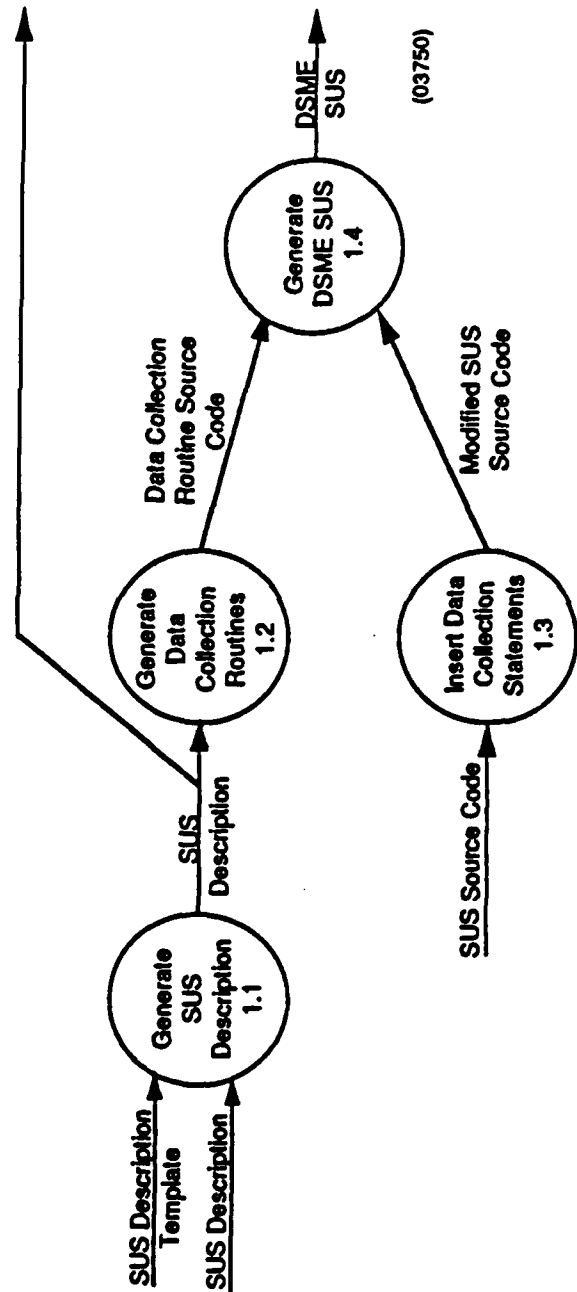


Figure 4-2, DSME DFD - Incorporate SUS (1)

Conduct Experimentation (2)

Conduct simulation experiments using baseline DSME subscriber simulation systems.

The function identifies the input and output components required for a simulation experiment, assembles these components and executes them to conduct the experiment, and analyzes the data collected during execution. A data-flow diagram of this function is provided in Figure 4-3.

Prepare Experiment (2.1)

This function is used to select, for later execution, the basic components of an experiment, including SUS, SUS data, and experiment analysis requirements. The function stores this data in an experiment configuration record for subsequent use or modification. A data-flow diagram of this function is provided in Figure 4-4.

Define SUS Configuration (2.1.1)

Specify the SUS and its required and optional components to be used for an experiment. A data-flow diagram of this function is provided in Figure 4-5.

Specify SUS Executable Components

Specify the main executable element of the SUS and any other associated modeling elements desired or required for execution.

Specify Input-Data Components

Specify the data elements which are required for the executable components to generate the desired simulation; these data elements might include entity descriptions and event descriptions or their parameters.

Specify SUS Control Components

Specify the control elements required for experiment execution.

Define Experiment Analysis Requirements (2.1.2)

This function specifies the analysis requirements of the experiment to indicate appropriate collection of data. Data collection is specified either directly, through identification of specific SUS collection elements, or indirectly, through abstract collection requirements or abstract report requirements.

Abstract specification is accomplished through specified abstraction hierarchies of elements, which permit aggregation of lower-level elements into classes of elements, and which are mapped to actual collectible elements in the SUS.

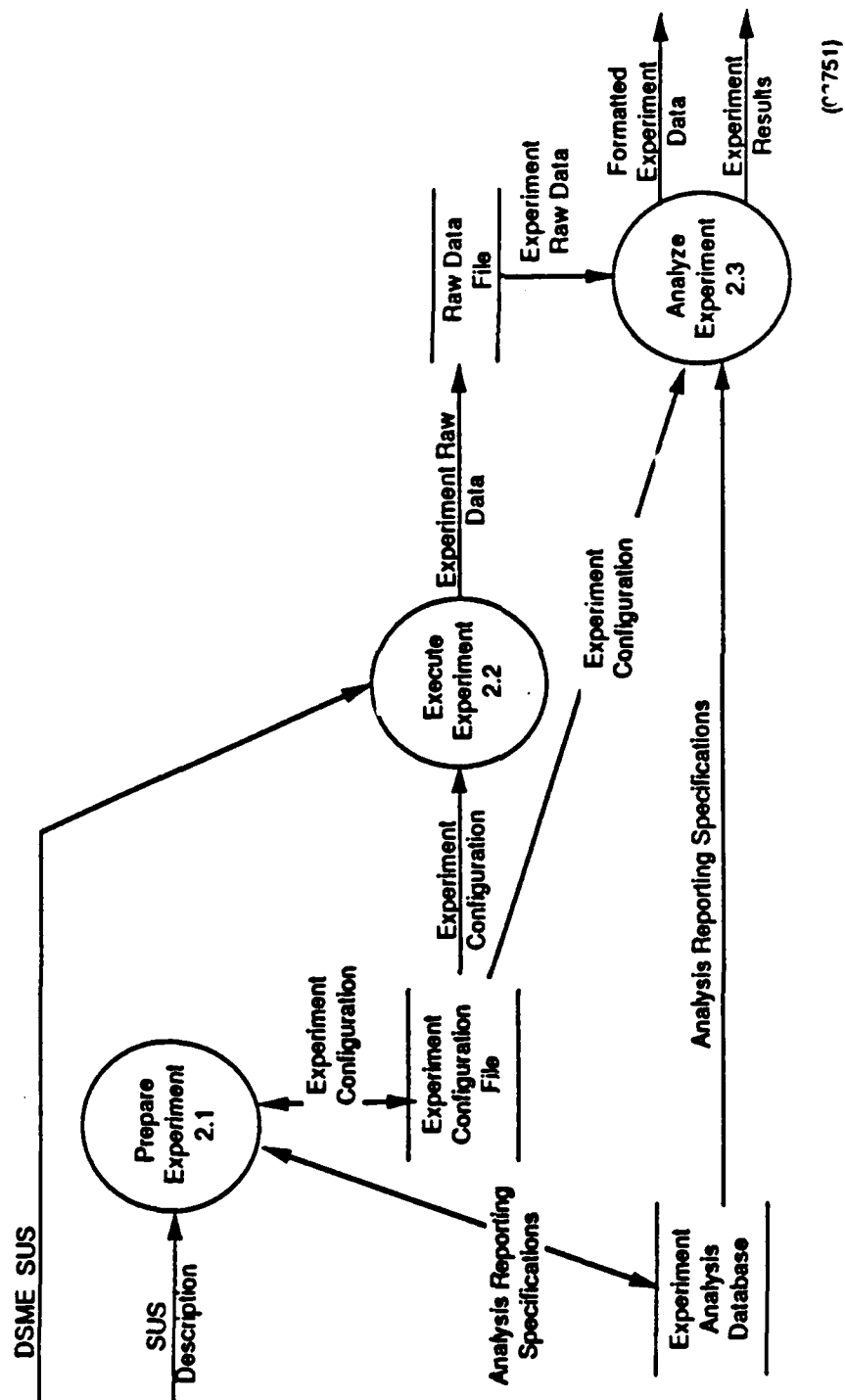


Figure 4-3, DSME DFD - Conduct Experimentation (2)

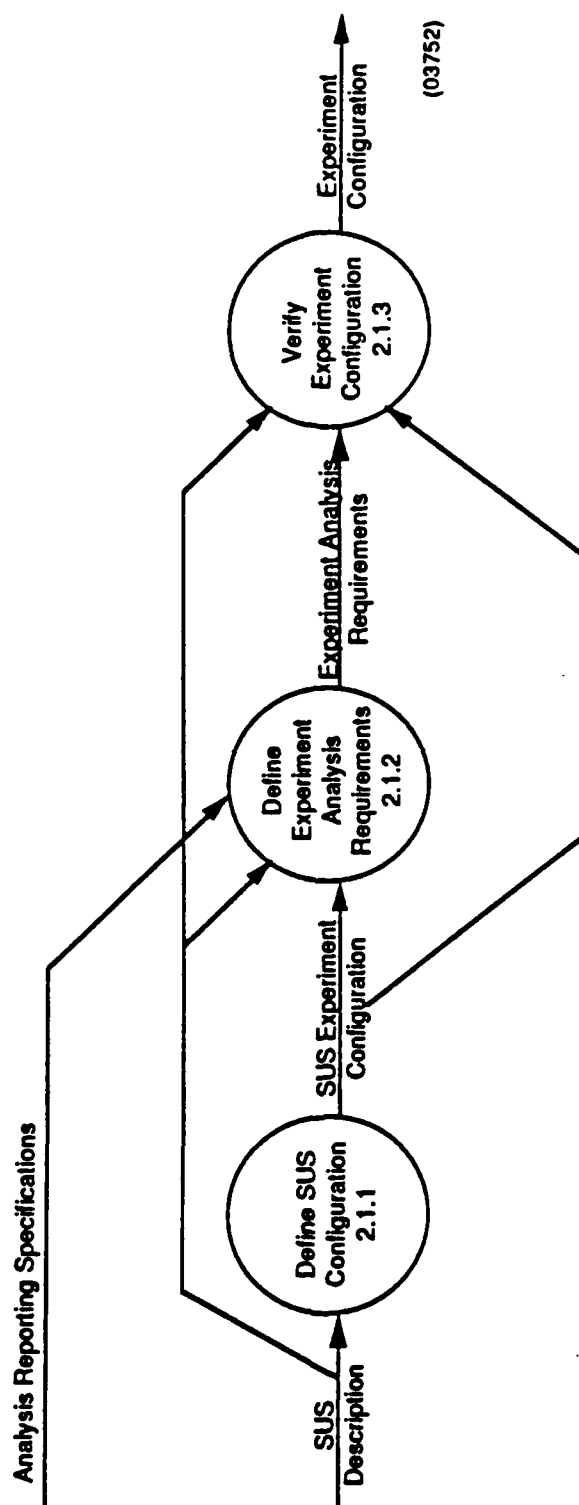


Figure 4-4, DSME DFD Prepare Experiment (2.1)

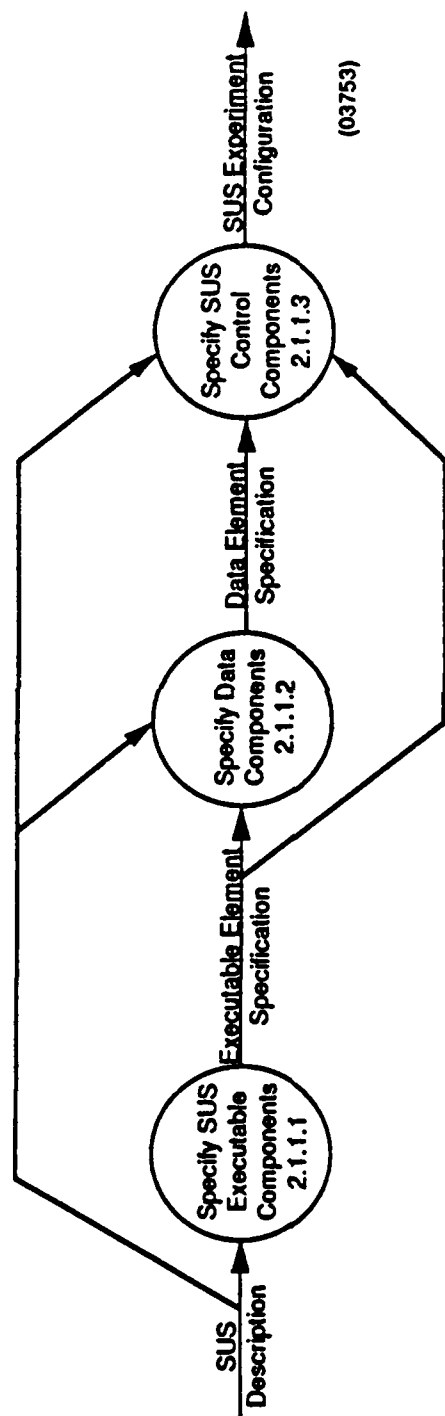


Figure 4-5, DSME DFD Define SUS Configuration (2.1.1)

A data-flow diagram of this function is provided in Figure 4-6.

Define Experiment Goals

Specify general experiment goals to assist the definition of general data-collection and analysis reporting requirements. A data-flow diagram of this function is provided in Figure 4-7. (Note that there will be need in the future to include higher-level descriptions of experiment goals than those provided by the following subfunctions.)

Select Data Collection Class

Select classes of data for collection from the analysis abstraction hierarchy (Data Abstraction Description). Classes are later translated into data-collection elements which are translated into data-collection requirements.

Select Data Collection Element

Select data elements for collection from the analysis abstraction hierarchy (Data Abstraction Description). Elements are later translated into data-collection requirements.

Select Reporting Requirements

Select report types from the experiment analysis abstraction hierarchy (Report Abstraction Description). Report types are later expanded into data-collection elements which are used to verify or specify data-collection requirements.

Translate Data Collection Requirements

Translate abstraction elements and classes to SUS-specific data-collection requirements.

Select Data Collection Requirement

Define specific data collection by indicating SUS-specific data for collection.

Verify Experiment Configuration (2.1.3)

Check consistency of experiment components based upon available DSME configuration management data.

Verify SUS Configuration

Verify the compatibility of the SUS components selected for the experiment.

Verify Experiment Analysis Configuration

Verify the validity and consistency of the data collection specified for the experiment.

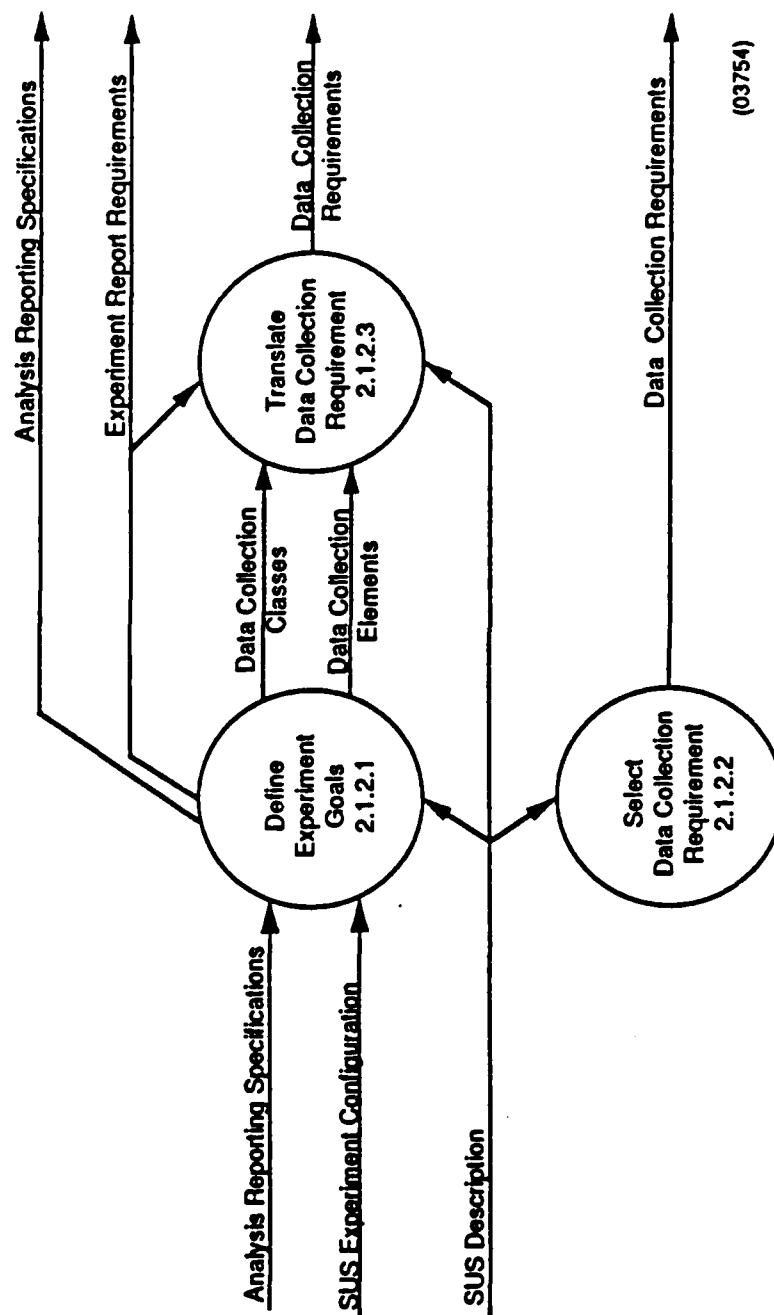


Figure 4-6, DSME DFD Define Experiment Analysis Requirements (2.1.2)

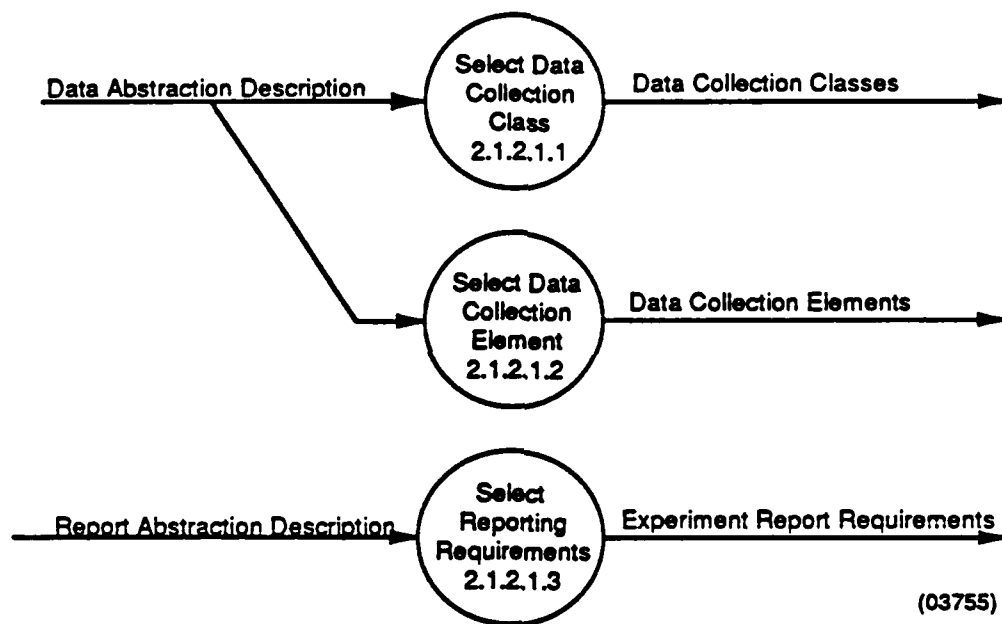


Figure 4-7, DSME DFD Define Experiment Goals (2.1.2.1)

Translate Experiment Configuration

Generate an experiment configuration object from the input SUS configuration and data analysis requirements.

Execute Experiment (2.2)

Execute an experiment according to the specifications of an experiment configuration record and analyst input. A data-flow diagram of this function is provided in Figure 4-8.

Configure Experiment (2.2.1)

Configure computing environment to execute the experiment by establishing logical connections to experiment components and by verifying the existence of the actual components. A data-flow diagram of this function is provided in Figure 4-9.

Assign Logical IO

Create logical assignments between designated experiment components and baseline DSME input/output devices.

Verify Experiment Configuration

Verify that the components associated with baseline DSME logical devices are accessible.

Translate Experiment Configuration

Generate control commands for the execution function from the experiment configuration object.

Run Experiment (2.2.2)

Execute the simulation experiment, monitoring and controlling its progress and collecting data for subsequent analysis. A data-flow diagram of this function is provided in Figure 4-10.

Control Experiment

Provide coordination and control for all baseline DSME run-time functions.

Execute Baseline DSME SUS

Conduct simulation modeling in accordance with experiment configuration under control of the Control Experiment function.

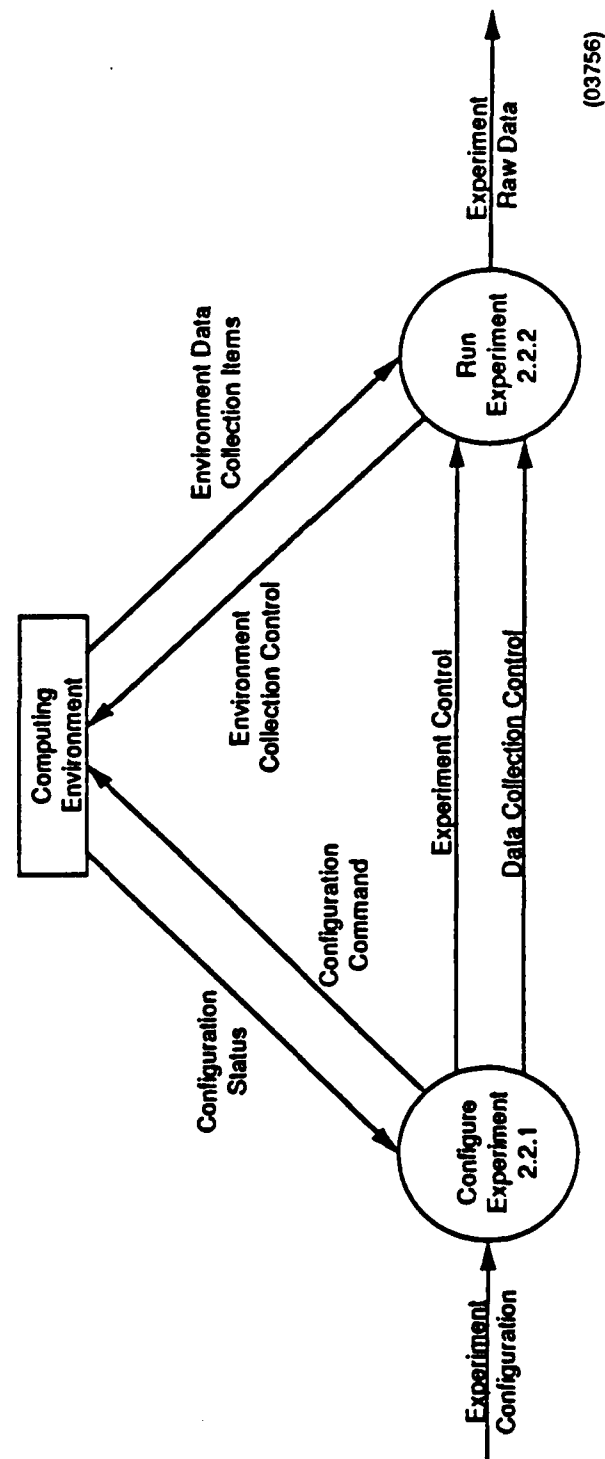
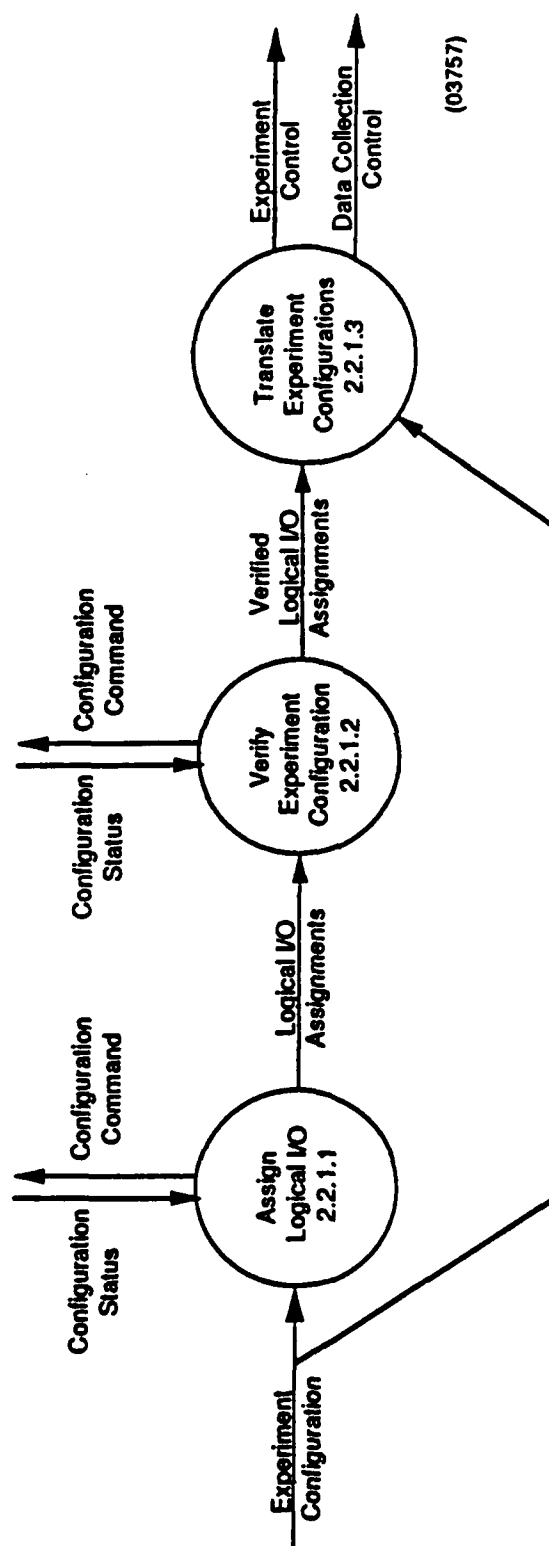


Figure 4-8, DSME DFD Execute Experiment (2.2)



(03757)

Figure 4-9, DSME DFD Configure Experiment (2.2.1)

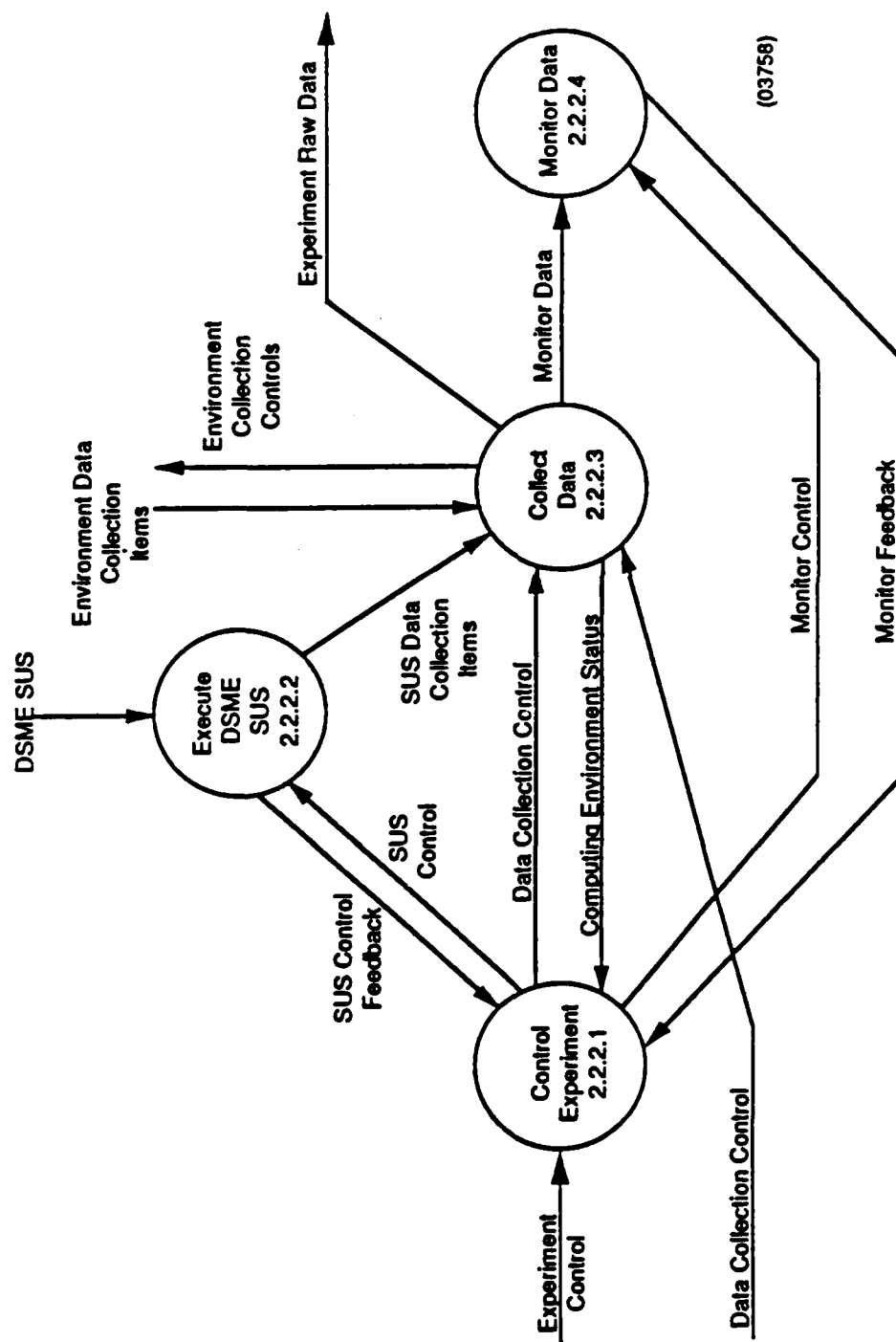


Figure 4-10, DSME DFD Run Experiment (2.2.2)

Collect Data

Determine action to be taken for each data-collection request, including time stamping, formatting, storing, and routing data items.

Format Data

Convert selected input data to output formats for monitor and storage.

Route Data

Route Formatted Data to appropriate destinations.

Monitor Data

Process data during experiment execution, initially to provide display services. Additional processing mechanisms could eventually be inserted to control the experimentation process or detect and handle errors.

Display Data

Present monitor data in selected screen formats.

Analyze Experiment (2.3)

Conduct analysis on experimental data by manipulating data, generating statistics, and generating graphical and tabular reports.

A data-flow diagram of this function is provided in Figure 4-11.

Organize Data (2.3.1)

Select, format, and load data into experiment database. The subfunctions of Organize Data are:

- Import Data
- Format Data
- Tag Data
- Filter Data.

Manipulate Data (2.3.2)

Manipulate data as determined by the analyst and the experiment configuration, including sorting, reformatting, indexing, exporting of data, and generation of statistical data. A data-flow diagram of this function is provided in Figure 4-12.

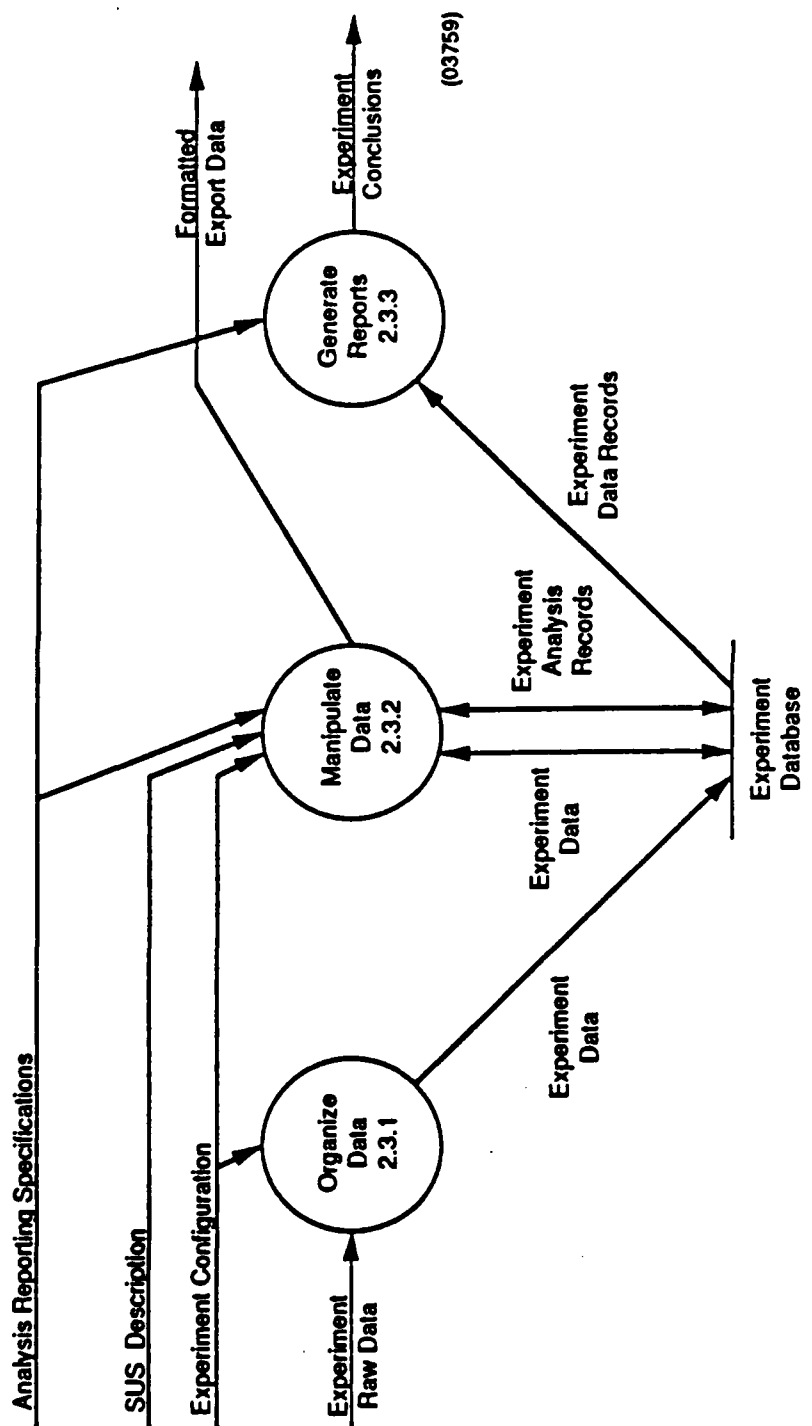


Figure 4-11, DSME DFD Analyze Experiment (2.3)

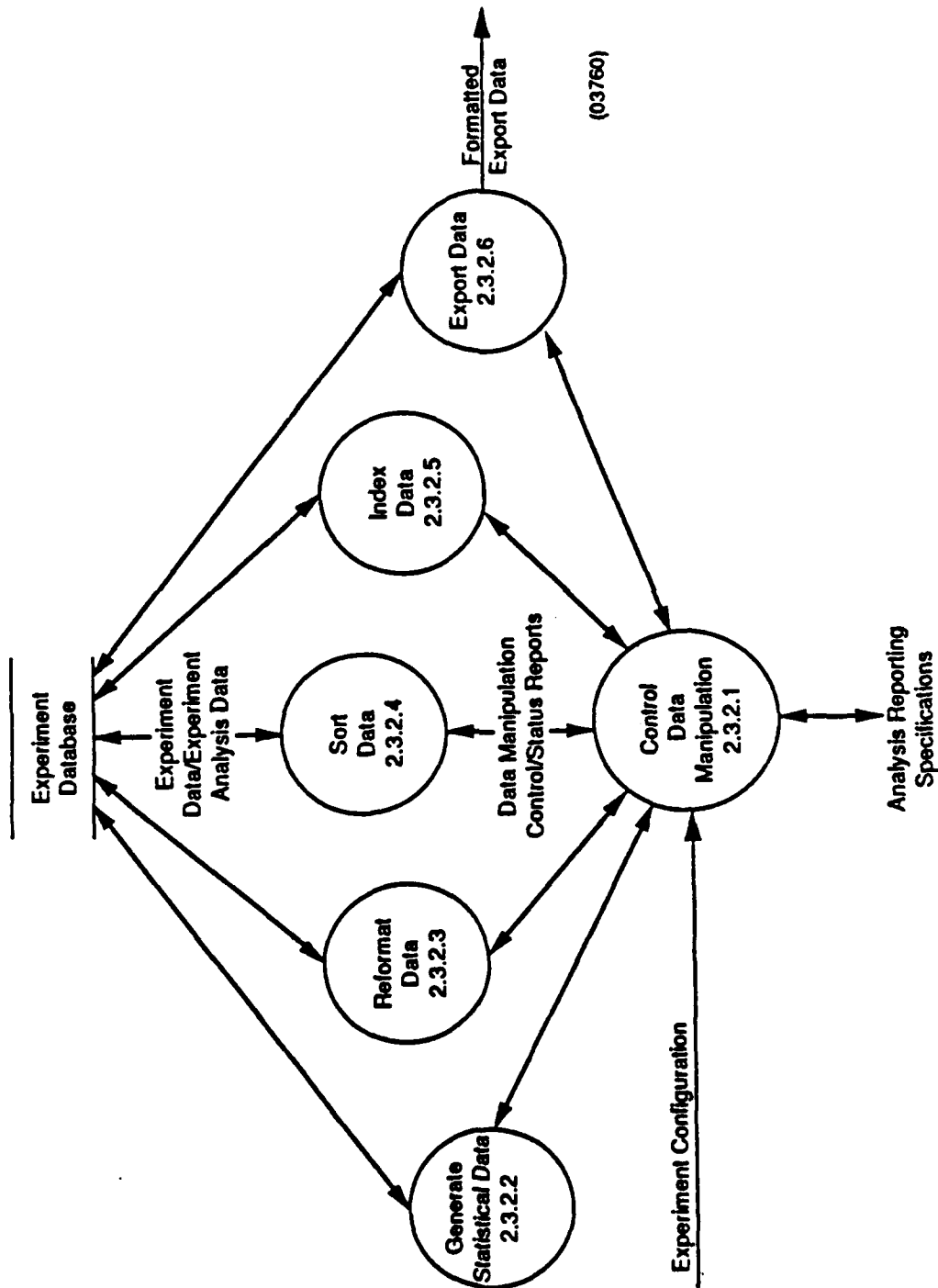


Figure 4-12, DSME DFD Manipulate Data (2.3.2)

Control Data Manipulation

Control and coordinate data manipulation functions as determined by the analyst and the experiment configuration.

Generate Statistical Data

Generate statistical data for collected experiment data.

Export Data

Format and store data in desired formats for external access.

Generate Reports (2.3.3)

Create tabular and graphical reports from experiment and statistical data. A data-flow diagram of this function is provided in Figure 4-13.

Translate Report Specification

Translates report specification to formats required by graphical-report and tabular-report generators.

Generate Tabular Reports

Create reports in the form of tables of data.

Generate Graphic Reports

Create reports in the form of graphs, bar charts, pie charts, etc.

4.1.3 Data Dictionary

Definitions for the terms used in the data-flow diagrams are provided in this section. Each term is defined below.

Analysis Reporting Specifications: An aggregation of the elements required to abstract the data collected during an experiment.

Computing Environment Status: Status reports sent from Collect Data to Control Experiment based upon results of environment data collection.

Configuration Management: Data required for management of DSME data and files.

Data Abstraction: Contains information relating to the data abstraction process for distributed systems. This abstraction is the decomposition of a distributed system in a hierarchy, permitting aggregation and generalization of system components. For example, a distributed system may be considered to be composed of the general subsystems of communications, processing, and control mechanisms.

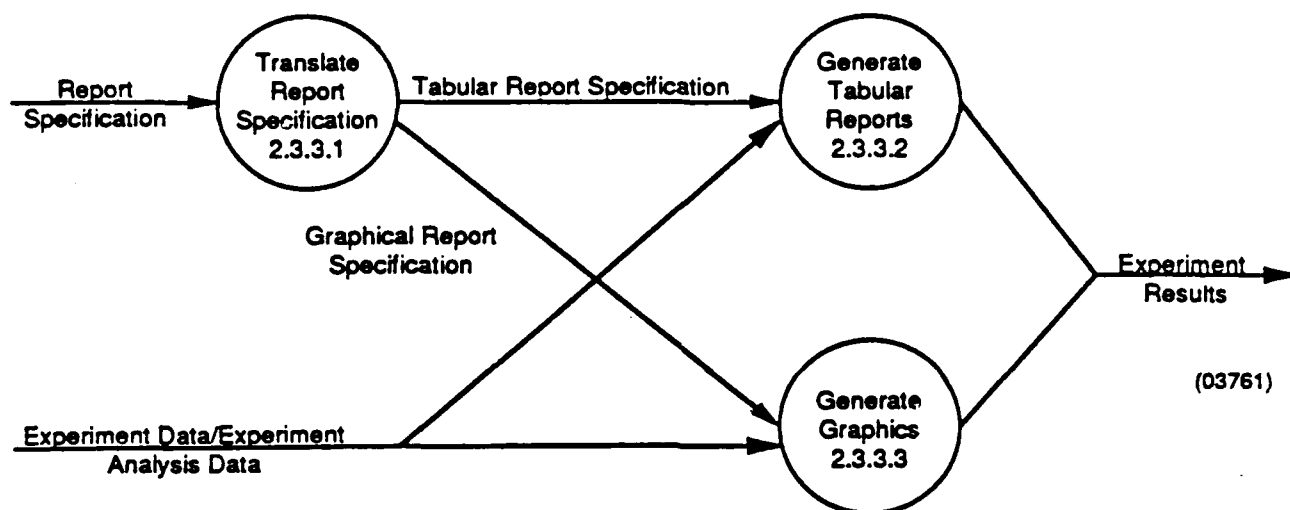


Figure 4-13, DSME DFD Generate Reports (2.3.3)

Data Abstraction Description: Description of the data abstraction relating distributed system (or tactical C³I for the SIM DRIVER test case) components, to be used in specifying general categories of data-collection elements for data collection. This is a description of the abstraction hierarchy for use by various DSME functions.

Data Collection Classes: Aggregations of data-collection elements within the distributed system abstraction.

Data Collection Control: Control for data-collection function, such as 'ADD COLLECTION' and 'STOP/START COLLECTION.'

Data Collection Elements: Collectible data elements within an abstraction of distributed systems and their components.

Data Collection Requirements: Specification of data collection in a format applicable directly to the selected SUS.

Data Collection Routine Source Code: Source code for DSME routines which are invoked by data-collection statements within the SUS.

DSME SUS: SUS adapted for use within the DSME.

Environment Collection Control: Control signals and requests for data sent to the computing environment (VAX/VMS).

Environment Collection Items: Data collected from the computing environment for use in SUS performance analysis.

Executable Element Specification: VMS Executable image and associated model specifications.

Experiment Analysis Data: Data related to experiment data, including results of statistics calculations.

Experiment Analysis Database: Contains support information for data analysis, such as report definitions.

Experiment Analysis Requirements: A description of the analysis which is to be conducted for a particular experiment, consisting of collection and report requirements.

Experiment Configuration: Description of all elements of a given experiment, including SUS, SUS components, data-collection, and analysis requirements.

Experiment Control: Control for experiment functions, such as 'STOP/START' and 'PAUSE.'

Experiment Data: Formatted data retained for analysis purposes.

Experiment Data Database: Contains all data collected for experimentation and information required for processing this data, including experiment configuration management data.

Experiment Raw Data: Formatted SUS Data Collection Items to be stored for analysis.

Experiment Report Requirements: Composition of data-collection elements in a report format.

Experiment Results: The results of an experiment consist of graphic and tabular reports.

Formatted Export Data: Experiment or experiment analysis data formatted for access by routines external to the DSME.

Modified SUS Source Code: Source code for SUS with data-collection statements inserted.

Monitor Control: Messages to control activities of the Monitor Data function.

Monitor Data: Data to be routed to the Monitor Data function for display or on-line analysis.

Monitor Feedback: Responses to Monitor Control messages.

Raw Data File: All collected data from an experiment.

Report Abstraction Description: Description of the data abstraction relating data-collection elements to specific reports and types of reports, to be used for data collection and report specification. The extension of the data abstraction hierarchy to the area of report generation.

SUS Control: Standard simulation control messages, such as 'PAUSE,' 'RESTART,' and 'STOP.'

SUS Control Feedback: Responses to SUS Control messages, including status and other data.

SUS Data Collection Items: Simulation Data generated by the SUS.

SUS Description: Contains complete description of SUS for use by various DSME functions. Description includes such items as required components, 'help' information, and attributes.

SUS Experiment Configuration: A description of the SUS components to be assembled and/or executed to conduct the experiment.

SUS Source Code: Source code for simulation under study (SUS)

4.2 ARCHITECTURAL CONSIDERATIONS

Given the preceding functional perspective of the system, we next address some specific architectural considerations for the design of the baseline DSME. The discussion begins with a general treatment of the software configuration for the baseline DSME (Section 4.2.1). Once this overall structure has been defined, the interface and the data passage between baseline DSME and a System Under Study (Section 4.2.2) are examined. A third issue that has broad implications for the design of the baseline DSME is that of experiment abstraction; this issue is discussed in Section 4.2.3. These first three subsections provide discussions of alternative approaches, and their advantages and disadvantages. Section 4.2.4 presents a recommended approach for the baseline DSME architectural design, based on the collectively most advantageous approach to accommodating the design considerations identified in the preceding subsections.

4.2.1 *General Software Configurations*

The baseline DSME will consist of three primary software components which perform the three basic steps of a simulation experiment—preparation, execution, and analysis—and a fourth component which assists the integration of a simulation system within the baseline DSME. The baseline DSME preparation component prepares the experiment for execution, identifying executable, data, and control elements for the simulation. The baseline DSME execution component is the component which collects data during the experiment in conjunction with the simulation software. The baseline DSME analysis component manipulates and interprets the collected modeling data. The final integration component provides some automated support in the configuration of a simulation for use with the baseline DSME, including baseline DSME database updates and required alterations of the simulation system.

Each of these software components is expected to execute independently, and will not require concurrent execution of any particular elements. This does not imply that the data flow between the elements is unimportant; for example, an analyst must still run the preparation software at least once before executing an experiment.

There are several possible configurations for the execution phase's two primary software subcomponents, baseline DSME and the simulation under study (SUS), shown graphically in Figure 4-14. For simplicity in the discussion of these configurations, each subcomponent is addressed as if it were a single process; it is understood that each may be implemented in a multiprocess configuration, and it is this overall implementation which is considered the subcomponent in the following summaries. These subcomponent configurations are:

- SUS linked to a baseline DSME main process;
- Baseline DSME linked to a SUS main process;
- Baseline DSME and SUS as concurrent, cooperating processes; and
- Baseline DSME and SUS as separate, nonconcurrent processes.

This section describes these configurations and some of their impact on the baseline DSME effort. In this discussion, the focus is on the data-collection function as the primary aspect of the baseline DSME. Also, much of the discussion of configuration impact will be postponed until more details of the system are defined in later sections, since this impact is most often related to design criteria.

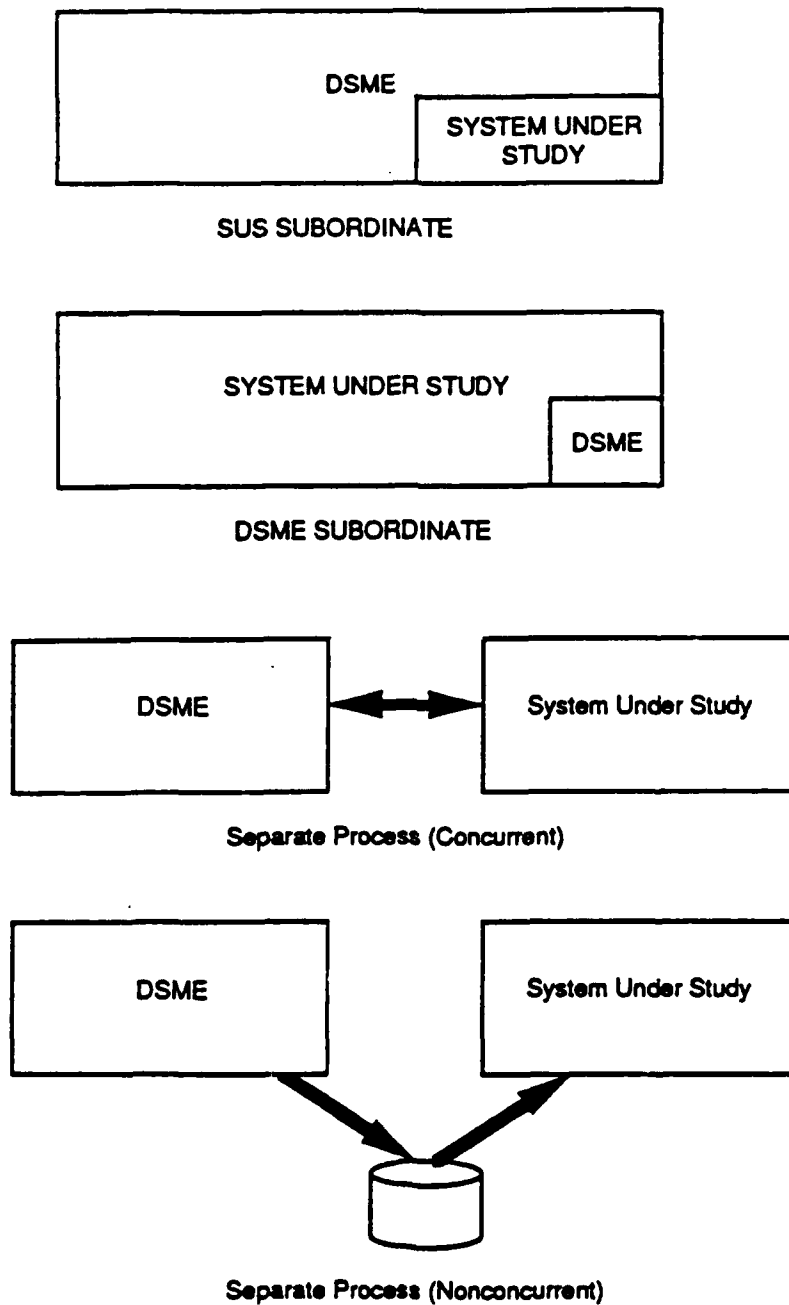
SUS Linked to Baseline DSME (SUS Subordinate)

The first configuration is based upon baseline DSME providing a run-time environment, within which the SUS is a (possibly dependent) component. The baseline DSME provides simulation functions such as experiment control and output display. This type of configuration directly supports an expansion of the baseline DSME's role, permitting evolution towards configurations where the baseline DSME provides more services. For example, baseline DSME could provide a SUS with such basic services as DBMS functions, making the SUS a dependent subscriber. Such configurations support the baseline DSME standardization goal which permits SUS developers to utilize existing baseline DSME tools rather than requiring them to develop their own, specialized versions.

The primary advantage of such a configuration is the standardization provided by the baseline DSME, which includes the baseline DSME standard interface and functionality. The user of multiple SUS will utilize the baseline DSME interface to work with each SUS, and will have a full set of baseline DSME tools to use with each SUS.

Baseline DSME Linked to SUS (Baseline DSME Subordinate)

The next configuration is the reverse of the previous, where the essential baseline DSME data-collection routines are linked with the SUS software to provide required services. In this case, the SUS normally remains independent, and could be executed without the baseline DSME routines. If the SUS is to become dependent on the baseline DSME functions, then configuration management becomes a major issue, since changes in the baseline DSME software implementing these functions may require changes in a number of existing SUS.



(03767)

Figure 4-14, Software Configuration

The primary advantage of such a configuration is the simplicity of the overall design and implementation. The primary issue of importance would be the formatting of data in data-collection routines, avoiding more complex interaction and interface problems inherent in certain other approaches.

Separate Process – Concurrent

The next configuration is one in which both software systems are executed simultaneously as cooperating processes, neither depending on the other. In such a configuration there will still be a requirement for at least a limited form of the above configurations, since some element of software must be implemented within either or both of the systems to permit their communication. The primary advantage of such a configuration is the power and flexibility provided by the individual processes which are essentially independent yet have access to the same data and resources.

Several communications mechanisms may be employed by cooperating processes in the VAX/VMS environment, as described in the following paragraphs:

File: This approach is based upon messages and data being written to and read from a common file. Such a mechanism requires that 1) the equivalent of a baseline DSME routine be implemented within the SUS to create a file of a specific format, or 2) that a standard baseline DSME meta-format, or format description, language be provided (which still requires a baseline DSME routine within the SUS). The basic difference between these two options is the susceptibility to changes in the baseline DSME; it appears that the second option is somewhat less susceptible to such changes, since data formats may be later interpreted by the baseline DSME despite actual collection changes. When such a mechanism is employed, a second design decision must be made about whether to use single or multiple files. Use of a single file requires more communications planning; use of multiple files results in more configuration management-related coordination.

Mailbox: The DEC VMS operating system provides a mailbox mechanism which supports interprocess communications. Mailboxes are objects which support queuing of incoming messages that are written by sending processes and may be read by receiving processes.

Network: DECnet network communications also provide implicit message queuing, thus supporting mailbox-like interprocess communications across processor boundaries. The DECnet mechanisms may also be utilized for single processor operations.

Separate Process – Nonconcurrent

The off-line configuration permits the SUS to be executed as desired, producing data which is at some later time processed by the baseline DSME. By its very nature, this approach requires the transfer of data by a file mechanism, described in the previous section. Again, this approach is not totally independent, since some routines within either the SUS or the baseline DSME must accommodate the formats of their partners, as

previously described in the independent run-time discussion.

The primary advantage of such a configuration is the simplicity of the design and implementation, which does not necessitate dealing with real-time interprocess communications.

4.2.2 General Data Collection Techniques

Three basic conceptual models are applicable to data collection – System Services, Direct Database, and Query Module. A graphic presentation of these models is provided in Figure 4-15. Each of these basic models has particular inherent performance characteristics which are of interest to the baseline DSME design and each may be implemented in various ways, further increasing the range of possible performance characteristics. Each basic model is discussed in this section, with particular emphasis on the advantages and disadvantages which are offered by the possible implementations of each. Table 4-1 provides a summary of advantages and disadvantages of each approach.

Table 4-1, Collection Method Summary

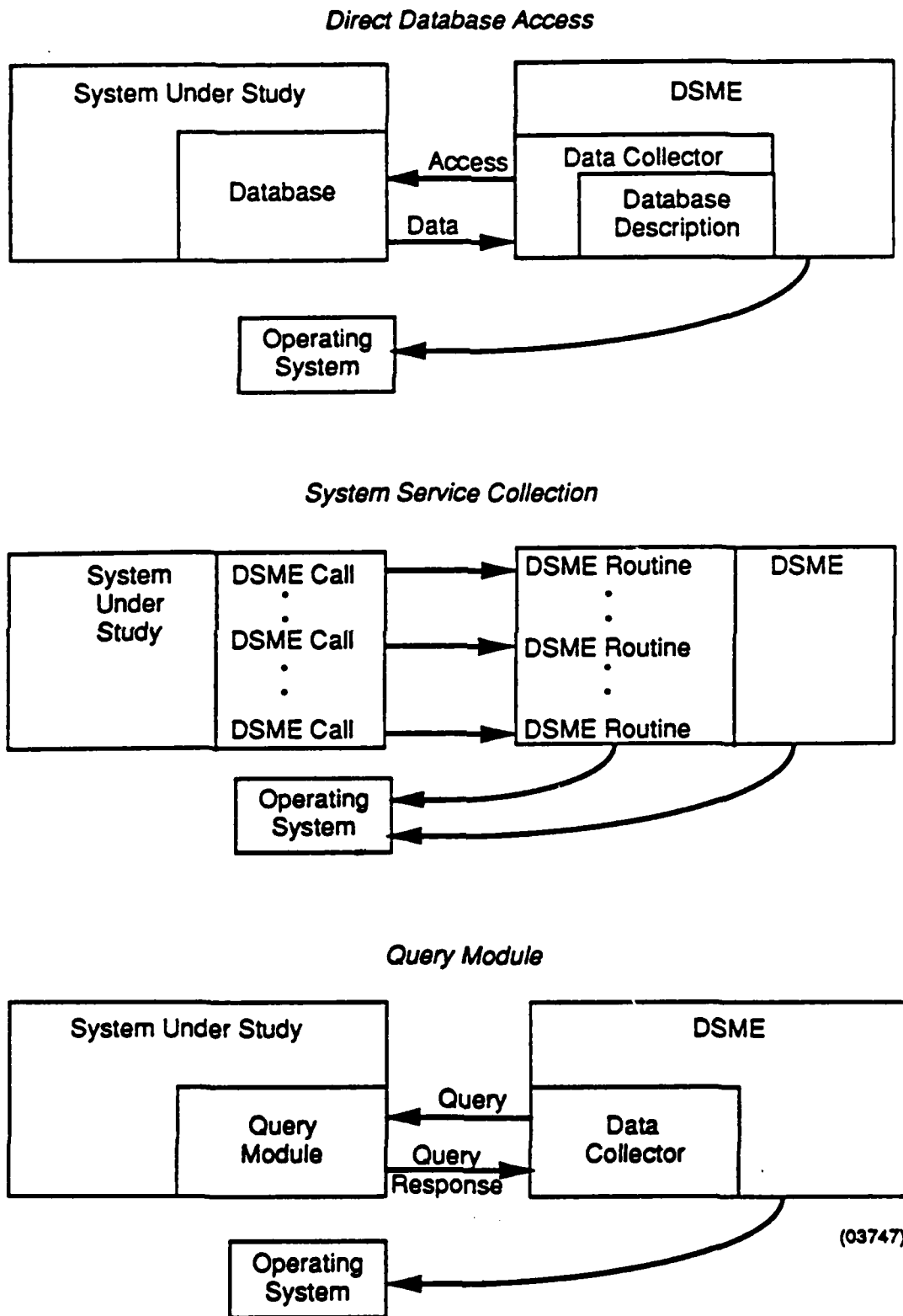
Action - Method	Service Call Approach	Direct Access Approach	Query Module Approach
Add New Collection Requirement	Modify SUS Code	Modify DSME code	Modify SUS/DSME code
Specify Collection Requirements	Set DSME flags	Set DSME flags	Set DSME flags
Collection Timing	Any	Periodic only	Periodic Only
Measurement Interference	Always	None	Only when collecting
SUS Change Impact	Limited	Significant	Limited
DSME Change Impact	Significant, boundable	Limited	Limited

4.2.2.1 System Services

The system service approach is based upon the insertion of calls to baseline DSME routines in SUS code. These calls would be in a format similar to:

```
DSME_Archive_Integer ( integer label, integer );
```

In an on-line configuration, the baseline DSME would combine timing information from other sources to provide a time stamp on the data and would then store the data in its selected format. In an off-line configuration such time stamping would be accomplished by the local collection routine.

**Figure 4-15, Collection Method Approaches**

Advantages

The primary advantages of the system service approach are:

Direct Linkage to Host SUS (Configuration Management): Since baseline DSME calls are included in the SUS source code, changes in data collection elements (variables) which affect these calls are detected at SUS compile or run time. This eliminates the more bulky procedures required to ensure compatibility in database formats required in the direct database approach.

Timing Control (Functionality): The ability to insert code statements wherever required in the SUS flow of control permits the collection of information at the occurrence of specific events, and permits the timing of events.

Supports Off-Line Operations (Ease of Use): The system service approach may be implemented in an off-line manner by writing baseline DSME calls or their results to a file which is later processed by baseline DSME. Either of these approaches would require some modification to the above sample baseline DSME call:

DSME_Archive_Integer (time stamp, integer label, integer);

A time stamp is required since without on-line access, baseline DSME does not otherwise have access to timing information.

Disadvantages

Linkage with SUS (User Interface): If a new data-collection requirement is defined for a SUS after initial design, additional baseline DSME calls must be made. The insertion of these baseline DSME subroutine calls within a simulation may require the recompilation and relinking of all or part of the SUS. The details of such a procedure are highly system specific, and the procedure does not seem a likely candidate for automation, at least at this point of system development. This means that a baseline DSME analyst will require knowledge of and access to the SUS code.

Dispersion of Baseline DSME calls (Configuration Management): The insertion of baseline DSME data-collection subroutine calls within the SUS widens the scope of the software affected by a change in the baseline DSME system. If such a baseline DSME call must be changed, various locations in the SUS may require alteration. Configuration impact can generally be limited by supporting existing baseline DSME subroutines throughout system evolution, precluding the need to frequently change original SUS baseline DSME calling statements. Naturally, only newly inserted statements would make use of new baseline DSME features.

4.2.2.2 Direct Database (Dataspace)

The direct database approach is based upon direct memory access to SUS internal data from the baseline DSME (the term database is used here to refer to the variables internal to the SUS required for simulation and control, and does not imply a more formal database such as the one defined in a DBMS). Data collection concepts are totally contained in the baseline DSME software and are used to control the retrieval of data elements from their mapped memory locations.

Advantages

Possible Indirect Linkage to SUS: A direct database access mechanism may be implemented in a data-driven manner, without directly accessing the internals of the SUS through a software linking process. For example, data locations may be identified through analysis of linker symbol tables requiring only the current versions of these tables to be accessed and combined with operating system loading information to permit access to data.

Data Collection Definition: Since direct access to the database is provided, definition of new data collection requirements only requires understanding of the data location, and does not require insertion of statements into SUS source code and recompilation. Therefore if the analyst determines the need to collect additional data elements, the SUS need not be modified, recompiled, and relinked; the data element must only be located in the SUS database mapping and collected using existing routines.

Efficiency: Since the external mechanism is directly accessing the database in memory, there is no SUS overhead associated with data collection. The only interference is the time-sharing load introduced by the baseline DSME access process. This is particularly important when the baseline DSME is utilized to examine SUS performance characteristics (rather than modeling characteristics).

Disadvantages

Timing Control: Both the direct-database and query-module approaches depend on the initiation of data collection calls from outside the SUS. Event-related initiation must therefore be based upon timing which is available outside the SUS, either through operating system functions or SUS messages. Some timing operations may be accomplished by directly accessing the simulation clock as a data element, providing the reference for other collection operations. An alternative is periodic initiation, whereby data is collected at regular intervals and integrated to provide certain other data requirements; this is frequently used in system monitoring. The periodic approach therefore may generate excessively large data volumes to achieve its goals, and does not appear feasible for many baseline DSME applications, such as event timing.

Linkage Complexity: Any means of determining the internal database structure of a SUS is extremely complex and subject to significant configuration management problems. One means for permitting direct database access is access to compiler/linker symbol tables

in combination with system loader information. Such a technique is highly dependent on a variety of system-specific software, and still requires more linkages between the automatically generated symbol table information and baseline DSME user interface symbol and class names.

Mechanisms

Several mechanisms might be employed to implement the direct database approach, including:

Separate Process Software: Assuming no linkage of baseline DSME and SUS software into a single executable image, extensive coordination of SUS internal data format must be accomplished prior to simulation execution.

Database Static Mapping: The simplest but least robust mechanism for direct database access is based upon a predefined mapping of the database format, relating data memory locations to standard baseline DSME data-collection elements, a mapping which is available to the baseline DSME software developer. This is less robust than more automated approaches since changes to the SUS database must also be incorporated in a new mapping which in turn must be incorporated in baseline DSME collection routines.

Access to Linker Information: Another mechanism for accessing SUS data involves interpretation of Linker symbol tables to determine names and locations of desired variables. When combined with loading information this permits direct examination of variable memory locations. Although this does permit automatic, direct access to the variables, some external mapping mechanism must still be provided to associate variables with standard collection categories, and configuration management is required when variable names are changed in the SUS. Also, this approach is much more difficult to apply to interpretive languages, involving on-line manipulation of a closed system, thus necessitating the development of another mechanism if these are to be employed for any SUS.

Database Communication (Dynamic Mapping): Another approach to accessing SUS data directly would depend on communication of database information from the SUS to baseline DSME collection routines at run-time. As part of SUS initialization, communications would be established with the baseline DSME software, and either predefined or all variable memory locations would be transferred for use in data collection. Such a mechanism is probably preferable to the other approaches, since it is simpler than the linker approach and less susceptible to SUS changes than the static mapping approach. However, it is not a simple approach.

Linked Software: If the SUS and baseline DSME software are to be somehow linked, partially or totally, the data access for direct database approach is simply implemented using such standard mechanisms as FORTRAN common blocks or Ada shared packages. Such a mechanism interferes only in the sense of competing for local processor resources: data access is simply a retrieval from system memory. Still complicated is the mechanism which permits the analyst to access the desired SUS variables without knowing the internals of the

system. Such a mechanism might involve a mapping of standard interface names to the SUS variable names, thus permitting specification which is essentially independent of the SUS.

Common Area/Package: When baseline DSME and SUS or their components are linked for execution, common data areas can be provided using certain programming language features. These features include shared Ada packages or FORTRAN common areas. This technique may not be possible for certain language combinations or special languages such as SIMSCRIPT.

DBMS: A final mechanism for direct database access depends on the utilization within the SUS of a standard DBMS for its data operations. In such a case, the DBMS provides an interface which may be used by baseline DSME (or any other concurrent process) to access required data, just as it is by the SUS. This mechanism does have one specific performance difference from the other direct database mechanisms, since it will compete with the SUS for data access due to limited access to the DBMS and due to the concurrency control established for the database contents.

4.2.2.3 Query Module

The query module approach is based upon baseline DSME access to SUS data through a query module incorporated in the SUS. The query module, linked within the SUS, exports a set of data-collection operations which it may accomplish. Baseline DSME collection routines would invoke these operations as desired to accomplish actual collection of SUS data. No other access to SUS data is provided.

Advantages

Direct Linkage to Host System (Configuration Management): Since baseline DSME calls are included in the SUS source code, changes in data collection elements (variables) which affect these calls are detected at SUS compile or run time. This eliminates the more bulky procedures required to ensure compatibility in database formats required in the direct database approach.

Well-Defined Linkage between Baseline DSME/SUS: The access to SUS data is basically object oriented in approach, since all access is defined by the exported calls of the query module.

Disadvantages

Timing Control: Both the direct database and query module approaches depend on the initiation of data-collection calls from outside the SUS. Event-related initiation must therefore be based upon timing which is available outside the SUS, either through operating system functions or SUS messages. The difficulties presented have been described in the summary of the direct database approach.

Flexibility: The query module method is based upon a well-defined interface definition for the SUS. Additions to that interface require corresponding programming changes to the query module.

4.2.2.4 Hybrid Approaches

It is also possible to combine features of the above approaches to provide particular performance advantages. In particular, the timing disadvantages of the direct database access and query module approaches will require some form of system service approach to conduct detailed timing studies.

4.2.3 Experimentation Abstraction

This section describes an overall abstraction for experimentation which will serve as the basis for the baseline DSME Data Collection and Analysis processes. Under this concept, experiment requirements are initially described from a high level of abstraction which is translated into levels of greater detail. This translation process provides the opportunity for standardization, aggregation, and verification. Standardization is provided by the generic definitions in a hierarchy of details. Aggregation is provided by that same hierarchy, where a reference to a single high-level abstraction also serves as a reference to a number of low-level components as defined by the hierarchy. Verification is a result of aggregation, since the explicit mapping (the hierarchy) can be used to verify user or system operations by serving as a reference for proper data structures.

It is believed that this approach has wide applicability in experimentation of all kinds, and is therefore not dependent on the baseline DSME nor is it limited to baseline DSME in its application. For this reason, the term SUS should be read in this section to mean system, rather than simulation, under study.

There exist two hierarchies, or trees, joined at their leaves, which comprise abstractions of data collection and data analysis.

4.2.3.1 Data Collection Abstraction

The Data Collection Abstraction is a decomposition based upon the entities and events of the SUS. The highest level of abstraction permits the specification of data collection in reference to various object or event types, described as data-collection classes. These data-collection classes are decomposed into specific data-collection elements, which are the leaves of this tree. Additional intervening class levels of decomposition may be appropriate or even necessary to achieve a complete representation, resulting in the creation of super classes. It should also be noted that the tree structures resulting from this decomposition are not necessarily distinct, since individual objects may be part of various classes.

Another aspect of this abstraction needed to specify data collection is some attribute to relate not only the object to be collected, but also the frequency at which the collection is

to occur. The basic collection frequencies are periodic and singular, with periodic collection occurring at some specified interval and singular occurring at a specified time. Event-based collection (e.g., collection which occurs when a specified event occurs, such as "when an object's state is changed" or "when an object is removed from the simulation") may be included as another category or as a special case of the above frequencies.

The exact relationships of the object and event abstractions are not yet fully determined. An initial approach would deal with a collection element as being an element defined by its two positions from each of the object and event hierarchies.

4.2.3.1.1 Data Collection Classes

Data collection classes are groupings of specific data-collection elements based upon object or event type and collection frequency. For a broad experiment goal such as "evaluate data communications," a variety of data collection classes may be specified, including: "communications links," "communications protocols," and "message traffic."

Data types: Data types are classifications of objects and events, such as communications objects, simulation events, simulation system events, and processor objects. Data types form a hierarchy of entities, with each lower-level type sharing the classification of its parent or parents. For example, hardware components include processor objects, which include cpu objects, which include cpu register objects.

Collection frequency: Collection frequency may be periodic or singular, and may depend on internal or external triggering mechanisms. Singular collection events are related to either a specific time (again either simulation or real), or more often are related to specific events. Examples of singular events are simulation start, event start, object modification, and user input. Periodic collection is scheduled based upon a simulation or real time interval, and is useful for database journaling, system profiling, and other events. Periodic collection is required when the occurrence of specific events cannot be directly accessed.

4.2.3.1.2 Data Collection Elements

Data collection elements are specific, normally-collectable data from within a data type which are specified for collection at a particular time or frequency. These data elements must further be mapped to specific elements of a SUS for actual data collection. For example, selection of the simulation system hardware environment cpu register object (mentioned above) can be mapped to a register dump collection technique at a specified periodicity, such as each simulation second or at the start of each simulation event.

4.2.3.2 Data Analysis Abstraction

The Data Analysis Abstraction is rooted upon Experiment Goals, which are decomposed into Summary Analyses, which are then decomposed into Reports. Reports are finally decomposed into data-collection elements, which are the leaves of the tree and

are identical to the data-collection elements of the data-collection abstraction tree. In this section it is easier to describe these components beginning with the leaves of the tree and progressing to the root.

4.2.3.2.1 Data Collection Elements

The data-collection elements of the analysis abstraction and the collection abstraction are shared, as indicated by the graphic of Figure 4-16. These elements are specific, normally-collectable data elements which are specified for collection at a particular time or frequency, and were described in the previous section.

4.2.3.2.2 Reports

A report is essentially a collection of data-collection elements, combined and presented in a specified manner. For example, a tabular report may be specified as a list of data elements collected at a specific time, iterated over a set of times. A graph may likewise be specified as a presentation of one or more related element values, such as size, versus another value, such as simulation time.

An example of this approach is provided by the General Research Corporation's GRCSIM tool, PIP. PIP provides a report facility which provides for user definition of tables and graphs. Tables are defined by a list of:

- data elements to be displayed,
- display formats for the data elements,
- column header text, and
- sorting direction.

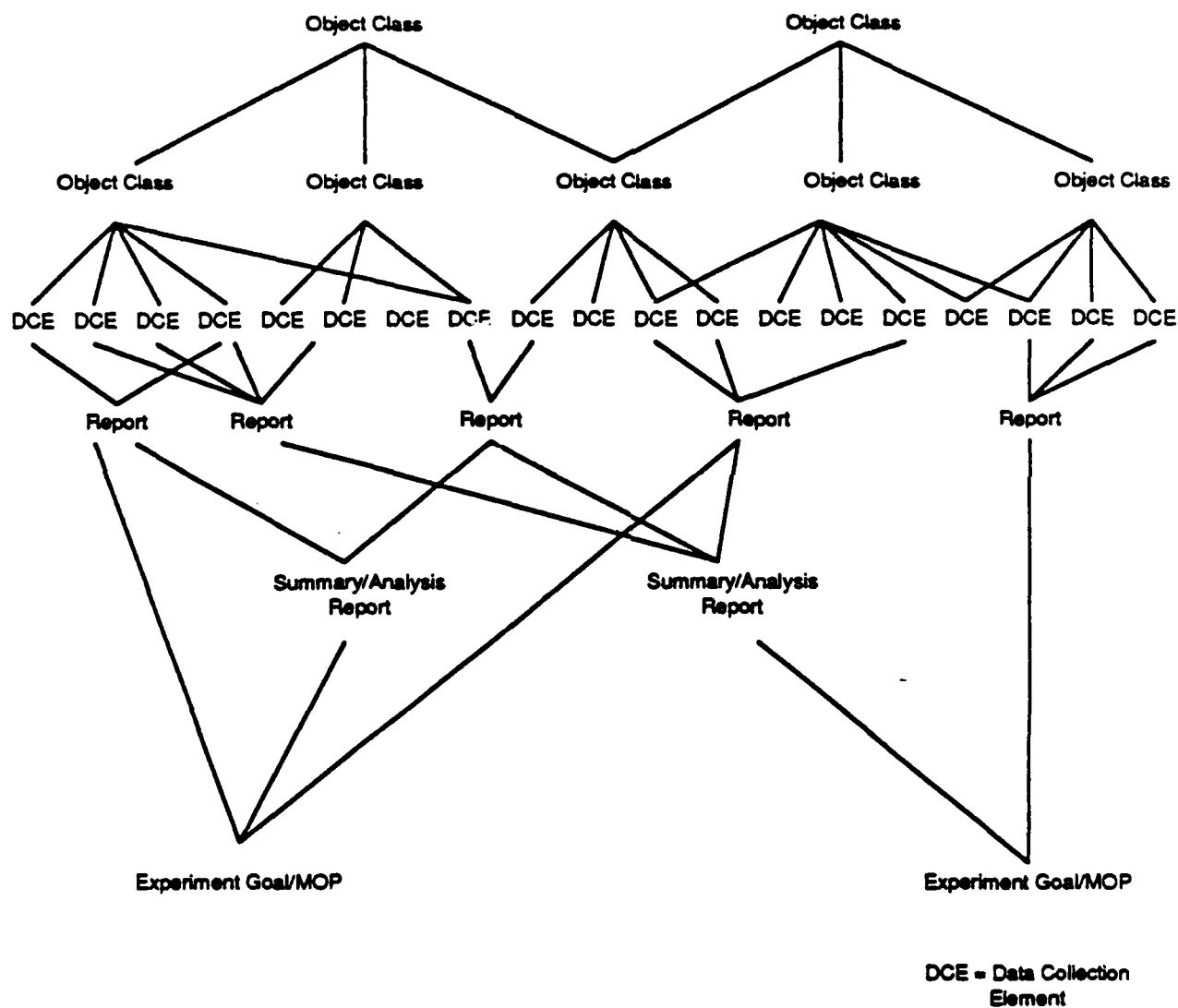
Graphs are similarly defined by ordinate and abscissa data elements in combination with display and formatting information.

4.2.3.2.3 Summary Analyses

The definition of Summary Analyses is somewhat weak at present, primarily because currently such analyses are normally accomplished under the direction of an analyst rather than automatically by software. An example of a summary analysis requirement would be the statistics which are generated for the columns of a tabular report, or the evaluation of a graph to determine its minima and maxima. Such intermediate results may be seen as similar to data classes in the collection abstraction, and are the result of further manipulation and "reporting" on reports.

4.2.3.2.4 Experiment Goals

Finally, overall experiment goals are a general description of experiment purpose, such as "evaluation of data communications," which are evaluated by analyzing the reports and summary analyses described above. Another view of these goals is that of measures of performance (MOPs) or measures of effectiveness (MOEs) of the SUS, since MOPs/MOEs



(03768)

Figure 4-16, Experimentation Abstraction

are generated as results of reports and summary analyses.

4.2.3.3 Application of the Experiment Abstraction

The general advantages of the Experiment Abstraction were summarized in the introduction to this section. By providing a well-defined structure which supports aggregation of its elements, an analyst is able to specify large numbers of detailed elements by specifying their class, saving significant effort and reducing the possibility of errors over techniques requiring element-by-element specification.

Not entirely clear are additional linkages provided by the abstractions. For example, analyst selection of an experiment goal or MOP relates at least indirectly to data classes to be collected, as well as to the specific data elements which comprise the reports and summaries associated with that goal. A mapping of collection classes to goals seems a worthwhile part of the abstraction for application to the baseline DSME.

The standardization concept is of particular interest to the baseline DSME through the development of these abstractions for the general case of distributed system and simulation analysis. If a general distributed system hierarchy can be developed, it will greatly improve the modeling of such systems in the baseline DSME by serving as a standard. Also, since analysis of the baseline DSME simulation systems themselves is necessary for debugging efforts and performance enhancements. A generalized hierarchy for simulation systems is also appropriate, and may also serve as a means of standardization. Such standardization permits the baseline DSME to better perform its role as a general environment for distributed system simulation.

The aggregation concept is useful in the development of a simple and effective user interface. By permitting the specification of perhaps large numbers of detailed elements through the selection of a single class of elements, the abstraction simplifies the need for analyst understanding and manipulation.

The verification concept is related to aggregation, since the reverse process is used. For example, if the user specifies an experiment goal of communications analysis, the data abstraction hierarchy may be used to determine if communications data elements are also requested for collection. If not, the collection may be made automatically or the user may be warned.

The abstraction of distributed systems is a concept which could greatly aid the standardization goals of the baseline DSME, since the mapping of particular SUS elements to a general abstraction will permit the use of the abstraction, rather than SUS details, in the user interface. Also, such an abstraction could lead directly to more standardized analysis, which is applicable not only to simulation results but also to the performance measurements taken on implemented distributed systems.

The abstraction of simulation systems is another important concept that will aid the functional and standardization goals of the baseline DSME. It is often necessary to evaluate the performance of a complex simulation system to provide a proper balance between its execution time and the level of its modeling detail. The concepts involved in this performance analysis are rather easily generalized, at least for particular classes of simulation (e.g., discrete event). For example, the details of event execution, such as the time spent in processing each event and the number of times the event was scheduled, are important analysis data. More closely related to simulations might be the evaluation of the functions related to the simulation system itself, such as event scheduling and database access. These types of data can be abstracted as described above.

4.2.4 *Recommended Approach - System Services*

In this section we describe a possible approach to the overall baseline DSME configuration. Based on the system service mechanism, this approach provides a high level of functionality with minimized technical risk. The basic configuration of this approach appears to PGSC to be the best orientation for the baseline DSME design.

The software configuration is based upon the separate-process, nonconcurrent approach previously described, with the SUS providing data to baseline DSME via file storage. Storage is accomplished by a collection of baseline DSME routines which are linked with the SUS and provide the translation between SUS and standard baseline DSME data formats.

The baseline DSME storage routines, coded in Ada, are called by each data-collection statement that is not protected by flags (e.g., not within the scope of an IF statement related to baseline DSME). The routines therefore perform some type of context checking to determine whether the collection is actually to be conducted, based upon the data element type and the current collection specification. This approach maintains control of data collection within a single, identifiable set of routines rather than depending on the probe inserter to prepare proper conditional clauses. Such an approach is feasible due to the low overhead of VAX/VMS procedure calls.

Data is stored in a number of data files, with each file associated with a collection class or a number of collection classes. This makes configuration management somewhat more difficult but reduces the number of data types in a particular file. This will also simplify the interface with the DBMS since the data relations in the database will generally be associated with data classes DBMS.

The data management functions of the data analysis phase will be accomplished using an interface to an existing DBMS. This permits the inclusion of a full range of data manipulation and storage capabilities rather than a subset which is implementable under contract restrictions. This also permits a simpler upgrade path when the DBMS is to be improved or replaced, since the developed interface will be more complete and more standard.

As indicated above, the data-collection and analysis abstractions will be applied in part to this effort. These abstractions will be modified to accommodate the special nature of the current effort's test case, and will be described in more detail under the following concept of operation section.

The differing advantages of the various design approaches have been detailed above. The System Services approach has been selected by PGSC for the initial baseline DSME implementation primarily due to a combination of design simplicity and its associated low risk. This is the most understood approach, from the viewpoints of both the concepts and the associated implementation details. The approach will provide significant capability to the baseline DSME, and will not preclude future modifications to accommodate alternative approaches.

Although recent investigations have indicated that the Direct Database approach may have particular performance and conceptual benefits, the details and feasibility of its implementation are not well understood. The effort required to validate this approach, combined with the possibility that it might not be realistically implementable, lead us to the conclusion that further investigation is necessary before committing to such an approach.

4.3 SUMMARY CONCEPT OF OPERATIONS

A description of the proposed baseline DSME concept of operations is vital to an understanding of the system and its associated technical issues. This section provides such a description, beginning with the three basic experiment functions of preparation, execution, and analysis. The discussion briefly describes the major actions required of the user and performed by the baseline DSME under each phase.

4.3.1 *SUS Incorporation into Baseline DSME*

Before being used with the baseline DSME there is a requirement for each SUS to be integrated with the baseline DSME system through a number of operations. For each SUS, a mapping of data-collection elements to SUS variables must be developed, collection routines must be inserted into the SUS to generate an output data file which is readable by the baseline DSME, and collection statements must be inserted in the SUS to actually implement the collection.

Programmer Operations

The following operations are accomplished prior to conducting experimentation by a programmer/analyst familiar with the SUS.

Insert Collection Statements: A programmer or analyst familiar with SUS code and baseline DSME programming conventions must insert collection statements at desired locations in the SUS, and if necessary recompile and relink the SUS. Other basic collection operations, such as event timing, could be inserted automatically. One method for automatic insertion would involve a search for the start of the the routine (such as the one

defined by the procedure header format for the implementation language), where a start timer statement is inserted, followed by a search for a procedure return where a stop timer statement is inserted.

Define Baseline DSME Data Interface: For each SUS it is necessary to create a mapping between the data collection elements and SUS data elements (variables). This mapping between names also requires specification of SUS data type and perhaps additional information. This information is required by the data-collection routines and by the data analysis routines.

Implement Collection Routines: This system service method will require that baseline DSME storage routines be programmed and inserted in the SUS executable code to permit the storage of data elements in a proper format for the baseline DSME. This programming task will be simplified by the implementation of a generic Ada storage routine which is simply instantiated for each data type to be collected.

Baseline DSME System Operations

The preceding section described those operations which could be accomplished automatically by baseline DSME system software; other than these, no specific functions which will definitely be implemented for SUS incorporation.

Prepare SUS Profile: A function which permits the user to define the execution configuration of the SUS will be implemented. This function prompts the programmer to input the commands required for SUS code initialization (such as compile and link commands), select and describe the required inputs for SUS execution, and specify the types of control parameters for the SUS.

4.3.2 Experiment Preparation

During preparation the analyst defines the models and data which will be combined to create a simulation run, and also defines the data which must be collected to record that run.

User Operations

The user of the system accomplishes the following operations during this phase of the experiment.

Specify Experiment components: The user must first identify the basic components of the simulation experiment, including at a minimum the simulation system to be used and the data collection to be accomplished. The baseline DSME will prompt for additional elements as they are required by particular simulation systems.

Select Simulation System: A simulation system must first be selected from a list of baseline DSME simulations. This selection determines which other elements must be specified to completely define the simulation run. These elements are also prompted and

may include model files, data files, and scenario order files.

Specify Data Collection: Although the simulation experiment will be executable after the previous specifications, output data must also be specified before the experiment is run. The specification of data collection is again related to the selected simulation system, but may be specified at least initially in a general manner for distributed systems experiments. Use of the data collection abstraction permits the analyst to specify all data collection at a high level, by selecting an experiment purpose. This purpose is then translated by the baseline DSME to data-collection classes which eventually translate to specific data-collection elements. Data collection may also be specified in a more specific manner by selecting options from the data class level, or may be specified in detail at the element level. Additionally, data collection may be specified from the output direction, by selection of analysis reports. The four functions internal to Specify Data Collection are therefore:

- Select Experiment Goals
- Select Data Collection Classes
- Select Data Collection Elements
- Select Output Reports.

Specify Experiment control: To complete the experiment preparation, the analyst selects the control options permitted by the baseline DSME and the SUS.

Modify (Edit) Existing Experiment: The analyst may additionally select an existing experiment specification from a system-provided list, and may modify the components of the specification to define a new experiment.

Baseline DSME System Operations

The following operations or functions are accomplished automatically by the baseline DSME system software during this phase of the experiment.

Support Simulation System Specification: As previously mentioned, the baseline DSME will be able to prompt the user for required input for experiment preparation. For each simulation system, baseline DSME will maintain a description of required experiment elements, such as executables, models, and data, and will also maintain a library of the existing elements to permit easy selection.

Generate Data Collection Requirements: If the user has selected data-collection options at the experiment goal or data class level, the baseline DSME software will translate these options to data collection elements. The data-collection abstraction hierarchy is the reference for this translation. The system then translates these elements to Data Collection Requirements for the SUS, as provided by the SUS mapping.

Verify Data collection: If the analyst has specified reports for the experiment, the system will verify that the data elements required for these reports have been specified in data-collection selection. If discrepancies exist, either the user is notified or the collection

elements are added automatically.

Verify Preparation: The entire experiment configuration is verified for completeness, as specified in SUS description database.

Store/Retrieve Configuration File: The complete experiment specification is stored in a text file which is used by the execution software to initiate and run the simulation experiment.

4.3.3 Experiment Execution

During experiment execution the analyst has some limited control of the experiment (as determined by the SUS) and may be able to monitor the data collection to support such control.

User Operations

The following operations are accomplished by the analyst user of the system during this phase of the experiment.

Select Configuration File: The analyst selects from existing configuration files (which fully describe an experiment by listing all required images and data). Although modifications will be permitted at this phase of the experiment, verification of modifications are only provided in the preparation phase software.

Control Experiment: The baseline DSME will provide a limited amount of experiment control depending on the control mechanisms supported by each SUS. It is expected that the analyst will be able to select the following control options from a menu: start, stop, pause, restart.

Monitor Experiment: Since data collection is accomplished by baseline DSME or baseline DSME-related software, it is reasonable to permit the redirection of such data to a monitor function which processes and displays selected data elements. For the current effort, it is expected that simulation time data will be the only data to be displayed.

System Operations

The following operations or functions are accomplished automatically by the baseline DSME system software during this phase of the experiment.

Initiate Experiment Execution: The baseline DSME software verifies the components specified in the experiment configuration file and then initiates execution of the experiment.

Collect Data: During experiment execution, baseline DSME routines within the SUS are invoked to store data elements. These routines accomplish initial formatting and processing (such as time stamping) of data, and also support storage and redirection (such as for experiment monitoring) of data.

Display Data: Baseline DSME data display routines accomplish the monitoring function as requested by the analyst. These routines accomplish the filtering and processing of the data for appropriate display.

4.3.4 Experiment Analysis

During the experiment analysis phase the analyst manipulates and interprets collected data using baseline DSME system tools.

User Operations

The following operations are accomplished by the analyst user of the system during this phase of the experiment.

Manipulate Data: The analyst is able to select data elements from the collected data for further processing, thus reducing the volume of data to be handled and stored. The analyst may also select data elements to be sorted and formatted for the most efficient storage and access.

Define Report: The analyst may define new reports using standard templates for each report type. The collection elements and report parameters must be specified, and may be recalled for future modification.

Select Report: The analyst may select a given report from a list of available reports. The initial version of the baseline DSME will provide primarily tabular reports, perhaps with some extremely limited graphics display capability (non-graphic bar charts). Additionally, limited statistical manipulation of tabular data will be provided. Statistics will include only: mean, median, mode, and standard deviation.

Tabular reports: The analyst may select from a listing of existing reports. Additionally, the analyst may define a new report and include it in the listing of existing reports.

System Operations

The following operations or functions are accomplished automatically by the baseline DSME system software during this phase of the experiment.

Determine Available Reports: The baseline DSME software determines the available reports for a given SUS by analyzing the data-collection elements from the experiment versus the requirements for existing report specifications. The resulting report list is used for user menu selections.

Create Report: A report creation utility supports the interactive creation of report formats for specified data. This report format is stored in a report definition format which can be reused or modified.

Conduct Standard Data Manipulation: For each experiment it will be necessary to process the sequential, mixed-type data which was collected. This manipulation processing will categorize, sort, and index the data as required for efficient access during subsequent analysis processing. As described under user operations, additional data reduction or manipulation may be conducted as requested by the analyst. Part of the standard manipulation package will be a component supporting the configuration management of experiment data files. This is necessary to maintain control over the variety of data files associated with each experiment, and to create simpler structures wherever possible. For example, selected data for a number of data classes may be incorporated in a single file for long-term data storage.

4.4 DESIGN CONCEPT APPLIED TO SIM DRIVER

The Simulation Driver Integration (SIM DRIVER) simulation system has been identified as the test case for the current DSME effort. SIM DRIVER is a discrete-event simulation system which was created by the integration of the TASRAN FORTRAN-based air surveillance simulation model with the Pascal-based DGTS simulation system and its associated battlefield models. The resulting simulation capability provided the functions of the original system in enhanced form, and supports their execution in a distributed DECnet VAX computing environment.

4.4.1 Test Case Discussion

The current test case for the effort will provide capabilities which support the overall goals of the DSME, but which are lacking in one respect. The specification of the Simulation Driver Integration simulation as the test case presents a generalized simulation capability on which DSME functions can be applied, but does not provide a simulation of distributed systems. Although models which simulate distributed system performance in a battlespace environment could be developed under the DGTS system (the basis of SIM DRIVER), no current plans for such development exist, and in any case could not be completed in the time period of this effort.

We must therefore describe the expected role of the test case in the DSME context, identifying which functions and structures will be tested by the SIM DRIVER application and which must be evaluated by other means.

4.4.2 Data Collection Abstraction

The goal of developing a generic distributed system data collection/analysis abstraction is not directly relevant to the SIM DRIVER test case. However, it is still useful to provide at least a limited basis for the distributed system abstraction to be used in future DSME developments, and a limited tactical C³I abstraction for testing of the DSME interface software with the SIM DRIVER simulation. Abbreviated outlines of portions of these abstractions are provided in this section.

SIM DRIVER Data Collection Abstraction (Tactical C³I)

It is possible to consider a data abstraction relevant to the SIM DRIVER tactical C³I simulation, involving a decomposition of the entities and actions relevant to such an environment. Such an abstraction, even in limited form, would be useful in testing the DSME using the SIM DRIVER simulation. Therefore, a restricted version of such a decomposition should be developed for the DSME testing task.

Simulation System Data Collection Abstraction

One abstraction which will be of use for the test case and for DSME future development is based upon the DSME role of evaluating simulation system performance. There are a number of general object (data element) and event classes which are applicable to all simulation systems or at least to all simulations of a particular type, such as discrete-event or process oriented categories. Summarized in outline form below is a sampling of this abstraction which is rooted in the experiment goal of "analyzing simulation system performance."

Analyze Simulation System Performance

Event Execution Class

Event count

Event time

Database Content Class

Database size

Database events (Event Execution Class)

Data retrieval

Data storage

Data modification

System Hardware Class

Processor

Disk

Memory

Communications

System Software Class

Operating System

Simulation System Software Class

Scheduler

Database Manager

Distributed System Data Collection Abstraction

The Data Collection abstraction is used in several ways to enhance this approach. First, provision of standard distributed system and simulation hierarchies supports a more standardized simulation interface. Second, these hierarchies are used to simplify user identification of data-collection requirements by permitting this identification at high levels

of abstraction. Third, the predefined hierarchical structure supports the verification and validation of user selections. Finally, the abstraction permits a mapping between the SUS and the hierarchies which permits automatic generation of the data-collection routines.

Hierarchy Description

Analyze Reliability

Analyze Communications

Communications Classes

Communications Hardware Class

Controllers

Media

Communications Software Class

Load Classes

Simulation message class.

Network control message class

4.5 OTHER ISSUES

There are a number of important technical issues which must be addressed under the baseline DSME concept, including:

- identification of the baseline DSME analysis capability context;
- determination of the potential for baseline DSME automated analysis;
- evaluation of performance measurement self-interference;
- determination of the validity of experiment stimulation;
- use of object orientation; and
- definition of target environment.

These items are summarized below (refer to Appendix D of the Functional Description for a more thorough discussion).

4.5.1 Analysis Function Context

One question of context is the approach to the analysis function of the baseline DSME. It has been determined that the analysis which is required to evaluate distributed system performance is essentially independent of whether that system is simulated or operational. For example, establishing the availability of a distributed system with particular operating parameters involves determining the time it was available versus the experiment time, and is not concerned with how that data is obtained. The data collected in the analysis of a simulation of a system is generated by the simulation model and reflects conditions in the model software. The data collected from an operational system is collected during system execution and reflects actual software and hardware state data. In either case, the desired data is similar or identical in nature, varying only in format, timing, accuracy, and/or validity.

The analysis of both sets of performance data is essentially identical, making the analysis function of the DSME a component separate from the simulation and monitoring (operational data collection) components. When dealing only with the baseline DSME, such a function is internal and need not involve consideration for the operational system analysis role, although this may still be appropriate to support long-term COTD objectives.

4.5.2 Analysis Implementation Potential

Another issue relating to analysis deals with the definition of analysis and the potential for meaningful, "high-level" analysis to be conducted. Automated analysis of distributed systems is difficult to generalize, since meaningful MOPs/MOE's are either not able to be generalized or are not even available for specific systems. What is currently feasible is the support of analysis, consisting primarily of the manipulation of data (e.g., sort, reduction), presentation of data in required ways (e.g., tabular report, graphics generation), and summarizing of data (e.g., statistics generation).

This ability may be enhanced by a capability to support the replication of analyst-selected combinations of the above operations. Basically a "batch" processing utility, such a capability permits replication across experiments, and simplifies the implementation of necessary modifications to such combinations. The batch processing could support the specification of data manipulation such as reduction, and presentations such as selected reports and graphs.

Future steps could evaluate the output from such low-level analysis routines to evaluate the system through automated MOP/MOE generation. This work would be at a highly theoretical level, dealing with the state of the art in distributed systems and their analysis.

4.5.3 Performance Measurement Self-Interference

Another technical issue involves the validity of simulation performance measurements despite the self-interference of the measurement system. Any software measurement procedure will involve some interruption of normal computer system processing to collect and store data. To accurately analyze the performance of the computer system the effects of the measurement processes must somehow be subtracted from the results.

Such a capability should be provided within the baseline DSME to support the optimization of simulation execution performance. This may be necessary to provide reasonable run times for complex simulations. Software techniques should be provided to monitor model and simulation system code and operating environment software and hardware.

4.5.4 Experiment Stimulation Requirement

The concept of stimulation, which is defined as the modification of the experiment while it is being conducted, is another issue. Although stimulation is accepted as a valid approach for experiments on operational systems, this is not necessarily so for simulation

experiments. Stimulation of an operational system is usually performed to artificially insert an event, such as node failure. Such alteration of an experiment can be planned and implemented as part of the simulation experiment; interactive modification during execution will alter the validity of the experiment concept. For example, removing modeled system load from a distributed system experiment would invalidate the original intention of the experiment.

One reason for altering an executing simulation would be to correct observed abnormal or other undesired conditions. Although this destroys the original experimental concept, it could save time by salvaging a run which would otherwise be worthless.

Another reason for run-time modification is the use of a simulation in an interactive role, such as personnel training or gaming. In such a case, no specific scenario is established for the experiment.

4.5.5 Use of Object Orientation

The concepts of object orientation have found particular application in the development of distributed operating systems and applications. Object-oriented design is also being applied in many simulation systems, and therefore must be considered as a factor within the baseline DSME context. The primary technical issues arising from the introduction of object orientation are the compatibility of any proposed baseline DSME approach with the object-oriented simulation systems of the future, and the benefits which may be obtained by designing baseline DSME in accordance with the concept which forms the basis for many operational systems.

4.5.6 DSME Target Environment

Another key issue that affects the overall design is the target environment within which DSME would ultimate operate. For the initial consideration, we assume the VAX environment at RADC; however, a capability as general as DSME ultimately should not be limited to a specific environment. Three aspects of the target environment must be addressed: 1) the simulation languages, 2) the simulation distribution, and 3) the simulation hardware/software operating environments which are considered applicable to the baseline DSME.

The issue of simulation language is pertinent since certain simulation languages, such as GPSS, are generally self-contained, including extensive data-collection and analysis utilities. Additionally, such languages and others may not permit the inclusion of separate data collection statements in model code, and may not permit other direct access to the simulation database.

The issue of simulation distribution is an interesting question in the baseline DSME context, and is important since the issues of simulation distribution involve the same issues as distributed systems in general, and introduce additional complexity as well. Distribution is becoming a more frequent technique as simulations grow in size and complexity, with

attendant increases in resource demands. Supporting this trend towards distribution is the increasing use of object orientation in simulation and other systems, which simplifies the distribution process.

The final issue, target hardware/software environment, involves the definition of the expected operating environment or environments for the baseline DSME. Although not critical to baseline DSME design, identification of a specific environment or number of environments can serve to focus or expand, as required, the orientation of the baseline DSME design. For example, if a VAX/VMS environment is identified as the only host architecture for baseline DSME, certain assumptions concerning the requirements of target simulations and the capabilities of the host system may be made.

5. DSME CONCEPT DEMONSTRATION SYSTEM

The actual concept demonstration system developed under this contract focused on the following concepts:

- a generalized, object-oriented user interface;
- experiment definition tools; and
- data collection from executing programs.

This section of the report documents the concept demonstration system.

This discussion, divided into seven subsections, begins by describing the demonstration experiment on which the DSME concept demonstration system is based. Next, Section 5.2 describes the concept demonstration system architecture. The remainder of the section is organized in the same manner as the structure of the system itself, with each subsection including a description of a component of the system. Section 5.3 includes a description of the User Interface,¹ which is followed by a description of the database created for the Concept Demonstration system using the User Interface (Section 5.4). Section 5.5 describes the Experiment Preparation component. While not part of the DSME concept, an integral part of the concept demonstration system is the System Under Study, the subject of the analysis supported by DSME. For the demonstration experiment, the System Under Study is the Simulation Driver Integration system, described in Section 5.6. The remaining two subsections describe the Loader component (Section 5.7) and the Reporter component (Section 5.8) of the DSME Concept Demonstration system.

5.1 DEMONSTRATION EXPERIMENT

The experiment selected for the Concept Demonstration System involves the collection of data from the execution of the Simulation Driver Integration system. The collected data includes both:

- functional/algorithmic data from the simulation model and
- execution performance data from the system.

The Simulation Driver Integration system is a simulation of a tactical Blue air surveillance network deployed against a Red airborne threat, developed by PGSC under a previous contract with RAD/COTD.

1. The User Interface represents the most significant amount of code written for the Concept Demonstration system; therefore more detail is provided in Section 5.3 than in sections describing other components of the DSME Concept Demonstration system.

This experiment was selected for several reasons. First, the Simulation Driver Integration system is an example of a distributed simulation capability; thus, it provided an example of both a simulation capability and a distributed system. Secondly, since the Simulation Driver Integration system had been developed recently for the same customer at RADC, it represented a continued thread of development within RADC/COTD. Thirdly, PGSC was also the prime contractor responsible for the Simulation Driver Integration development, and therefore was already intimately familiar with the detailed internal workings of the system. This allowed PGSC maximum concentration to be focused on developing the concept demonstration software and experiment, rather than on learning a System Under Study.

5.2 CONCEPT DEMONSTRATION SYSTEM ARCHITECTURE

Figure 5-1 shows the architecture for the Concept Demonstration system. Each of the components is introduced below and described in more detail in a subsequent subsection.

The User Interface, which is the largest of the components, provides an object-oriented interface for the user to populate the DSME database. The DSME database, implemented in the RIM-5 Database Management System (DBMS), contains information about the System Under Study (SUS), which for the demonstration system is the Simulation Driver Integration system. Such information includes the data files required to execute Simulation Driver Integration as well as different types of experiments that can be performed.

Given an experiment name from the user, the Experiment Preparation component accesses the DSME database, extracts the information necessary to execute that experiment, and verifies that the files needed to run the experiment exist on the system. The extracted information is used to create the Experiment Script, which is actually a VMS command file that can be executed to run the experiment.

The DSME Experiment Execution component runs a specified experiment script. In the DSME Demonstration System, the Experiment Script includes the command to run Simulation Driver Integration, along with the names of the model file, the initial orders file, and the entity data command file to be used for that particular test run. The user is also prompted to enter probe collection information (i.e., when to collect probe data and what type of probe data to collect) which is then collected during the execution of the experiment and is stored in the DSME Dump database. Also stored in the DSME Dump database is system performance information.

As previously indicated, the System Under Study is the Simulation Driver Integration system. It was created by interfacing a tactical battlefield activity simulation model (the Dynamic Ground Target Simulator, or DGTS) and a model of Blue surveillance assets such as TPS-43 ground radars and AWACS aircraft (the Tactical Air Surveillance Radar Netting, or TASRAN) model. The resultant capability maintained the integrity of the individual models, while defining a message-passing protocol ensuring that information in the

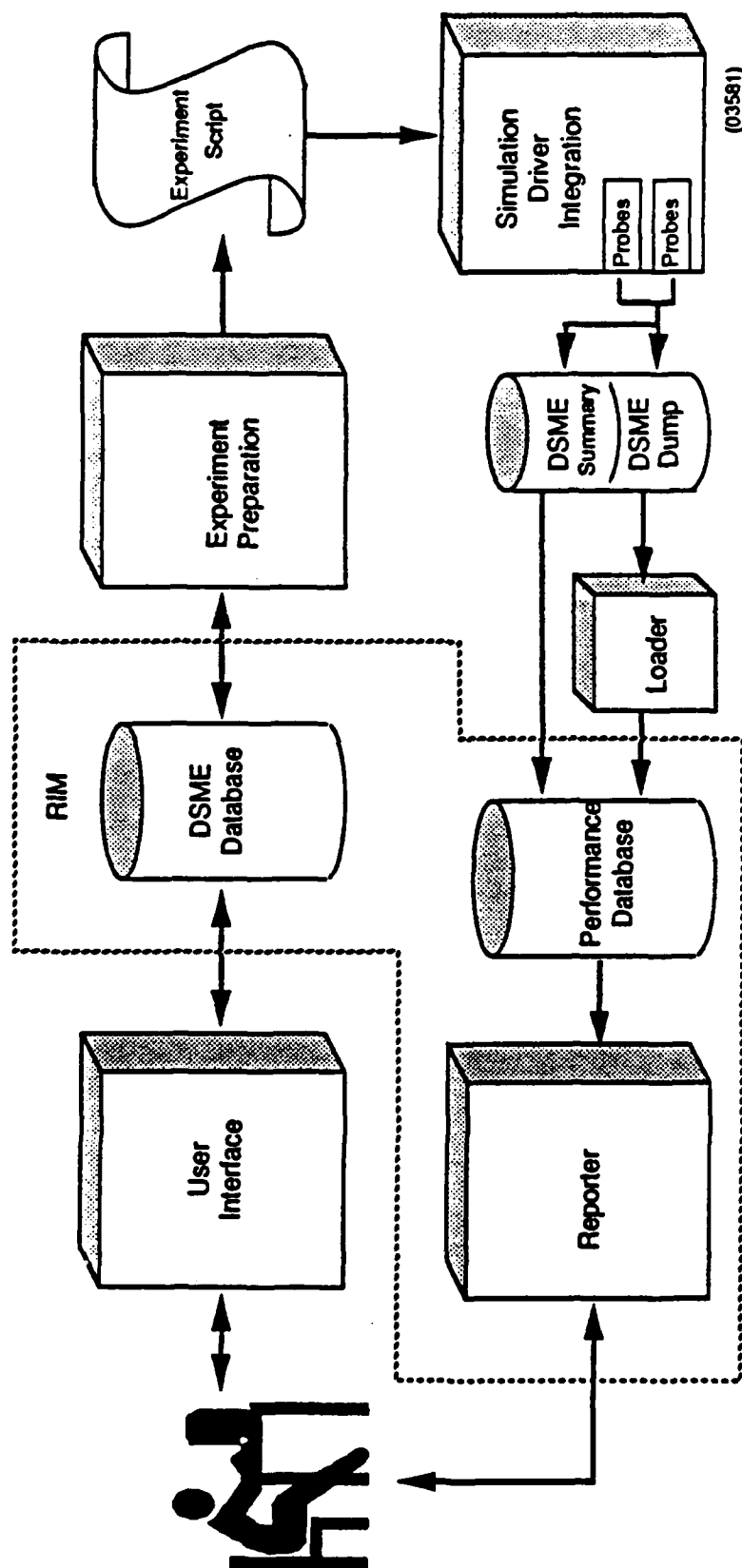


Figure 5-1, DSME Concept Demonstration Architecture

component databases remained consistent. Synchronization in the form of a rendezvous occurs at the beginning of each simulation second. The individual models are defined as separate processes with DECnet message passing, so that they can both reside on the same processor, or they can reside on different processors connected via DECnet.

The Loader takes data generated by the probes inserted into the Simulation Driver Integration system and loads the probe data into a RIM-5 database. The generated probe data is initially stored in the DSME Dump database, therefore, the Loader must reformat the data into RIM database loading commands and then use the load commands to create a probe database. The data in the DSME Dump database is in two forms: 1) a binary file containing scenario object (vehicles, units, aircraft, sensors, etc.) information, and 2) a text file containing scenario performance information. The Loader extracts the data from the binary file and the text file, and creates load statements in RIM format. The data is then loaded into the RIM database for subsequent manipulation. This data is called the Performance Database.

The final component, the Reporter, is actually a set of RIM statements for extracting and formatting data in a RIM database. The Reporter includes a specific set of commands for formatting and presenting data that has been stored in the Performance Database. There are several predefined reports that may be created concerning the data within the Performance Database. These reports are presented to the user in menu form and the user may select one or more reports to be created. These reports are output in text format for the user to view either by using a text editor or by printing.

For a computing environment, the implementation was limited to tools that execute on Digital Equipment Corporation's (DEC) VAX series of minicomputers running under the DEC VMS operating system. System communications were provided by Ethernet-based DECnet protocols and equipment.

5.3 USER INTERFACE COMPONENT

The DSME User Interface allows the user to define a System Under Study (SUS), using an interactive, object-oriented system. When defining the SUS, the user establishes relationships between objects, attributes describing each object, and conditions under which certain actions will take place. Currently, the only SUS that has been defined via the User Interface is the Simulation Driver Integration system. Once the SUS has been defined, the information is stored in a RIM Database and is then used by the other components of the DSME Demonstration system to run experiments and analyze the results. This section describes the DSME User Interface and the procedures and functions that comprise the system. The following paragraphs provide background in understanding the reasons behind the selection of both the graphics package and the Database Management System (DBMS).

An important requirement in the development of the DSME User Interface was portability. In the graphics area, this requirement implied separating graphics routines from other parts of the system, so that a new graphics package could easily replace the current

graphics package in the future. This requirement resulted in the allocation of all graphics routines to a single package, named *Graphics*.

The choice of the graphics approach used in the DSME User Interface was influenced by low cost, and made at the expense of graphics performance. This choice was considered acceptable because the delivered DSME Concept Demonstration system is a working prototype and not an operational system. Much of the functionality in the graphics package (e.g., multiple windows and scrolling) is provided as an integral part of many current graphics packages and workstation devices and will be unnecessary in the application code of an operational DSME system. Such an implementation will also significantly increase performance.

Along with a separate package to contain all graphics routines, there is also a single package, *Database Interface*, that contains all the database calls to RIM (the DBMS used in the User Interface). This package centralizes all the Ada/FORTRAN interface routines and can therefore easily be replaced if a different database package is used in the future.

As with graphics, the choice of a database resulted from a design tradeoff that reduced cost at the expense of performance. Implementing the DSME User Interface with a commercial database such as ORACLE will significantly improve performance and make the FORTRAN routines obsolete through the use of Ada/SQL interface routines.

Data flow within the User Interface is shown in Figure 5-2. The Executive Software establishes the file name for a user's list of available databases available (if any) and passes this file name to the System Specific Tasks unit which creates or opens this file. The Database software component interacts with the database by sending or retrieving data. This data is passed from or to the Objects unit. The Database software component also has to send file names of new or old databases to the System Specific Tasks unit to be created or opened. The Objects unit passes values to the Attributes unit and actions to the Scripts unit. It also passes various data to the Tools unit. The Graphics component receives parameters from the user and displays data for the user via the terminal screen. It also passes screen locations to the location trees to search for the corresponding record with that location. The object ID is then retrieved from this record and passed to the Objects unit where the appropriate object is found.

The global data represented below actually is the set of common data structures used for the Distributed System Modeling Environment (DSME). Due to the nature of the Ada language, no true global data is actually present in systems implemented in Ada; rather, data structures are part of packages accessed by other packages through a set of functions and procedures.

The common data structures for DSME start with an object record, the main structure of the system, which has an attribute list, script list, and relation list associated with it. All object records are stored in an object tree in alphabetical order. Parent/child relationships between objects are maintained by pointers. Location trees that associate an object with a

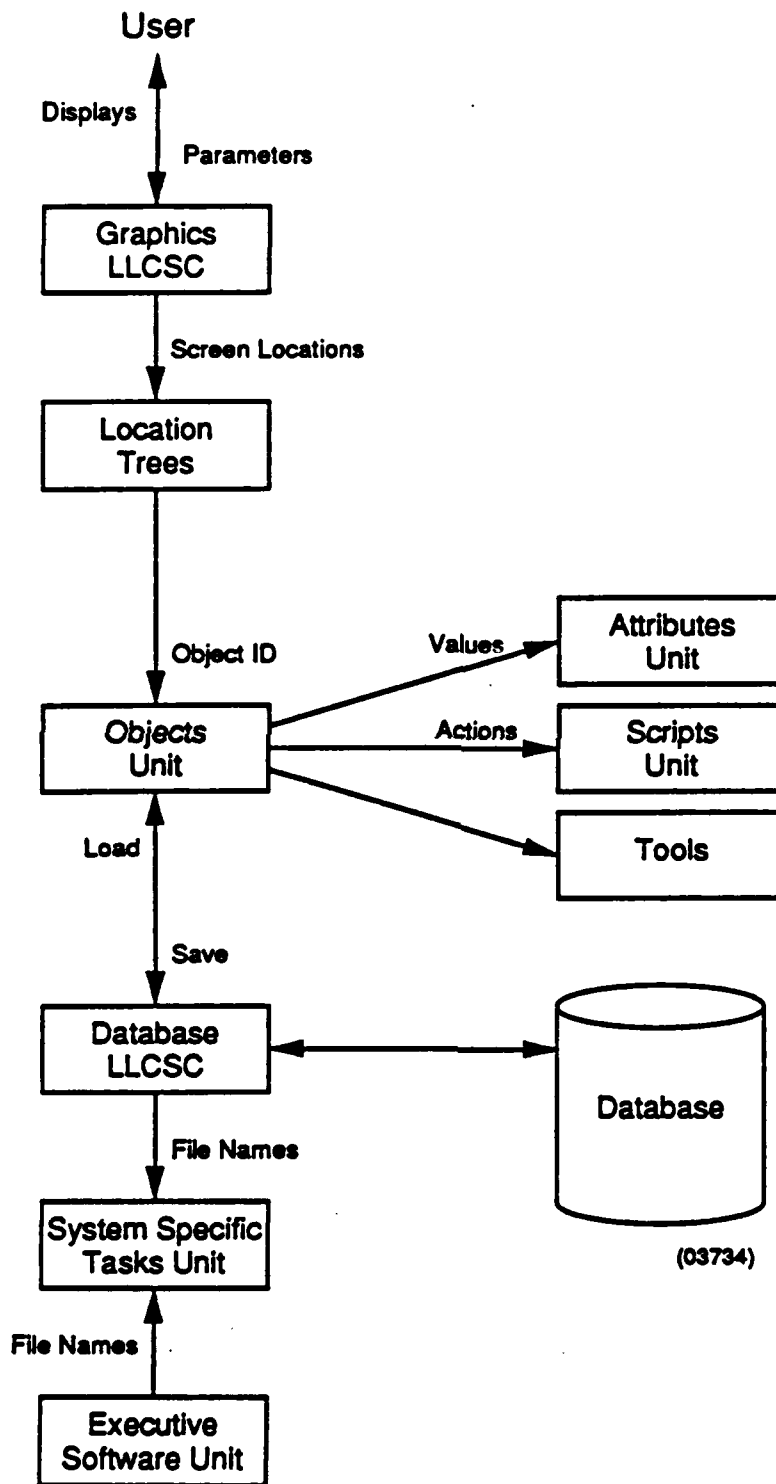


Figure 5-2, CSCI Top Level Data Flow

location on the screen are maintained. The records stored in these location trees contain screen location information and a pointer to the associated object record in the object tree. A diagram of the overall data structures is shown in Figure 5-3.

The overall screen format and a summary of the available commands are presented in a series of figures and tables. Figure 5-4 shows the Genealogy screen with all of its commands under the various pull-down menus. This screen primarily loads or saves genealogy structures and provides commands to create new genealogy structures. Table 5-1 gives a short definition for each of the possible commands and command abbreviations in the Genealogy screen. The letters within the parentheses are the command abbreviations that the user may enter instead of selecting the menu item by using the arrow keys.

Figure 5-5 shows the Attributes screen with all its commands under the various pull-down menus. This screen primarily defines objects by providing commands to establish attributes for objects. Table 5-2 gives a short definition for each of the possible commands and command abbreviations in the Attributes screen.

Figure 5-6 shows the Relationships screen with all its commands under the various pull-down menus. This screen primarily relates existing objects in the genealogy structure by providing commands to define relationships and structures for objects. Table 5-3 gives a short definition for each of the possible commands and command abbreviations in the Relationships screen.

Figure 5-7 shows the Scripts screen with all its commands under the various pull-down menus. This screen primarily establishes scripts and actions that the scripts should execute for objects. Table 5-4 gives a short definition for each of the possible commands and command abbreviations in the Scripts screen.

The final aspect of the User Interface described herein is the structure of the software that accomplishes the functionality identified in the preceding paragraphs. This information is provided to establish what was actually accomplished as part of the DSME effort. More detailed information about the software packages can be found in the *DSME Program Maintenance Manual*. The DSME User Interface is separated into several different packages:

- Database Interface,
- Objects,
- Attributes,
- Scripts,
- Location Trees,
- Scr Const,
- Tools,
- Parser,
- Actions, and
- System Specific Tasks.

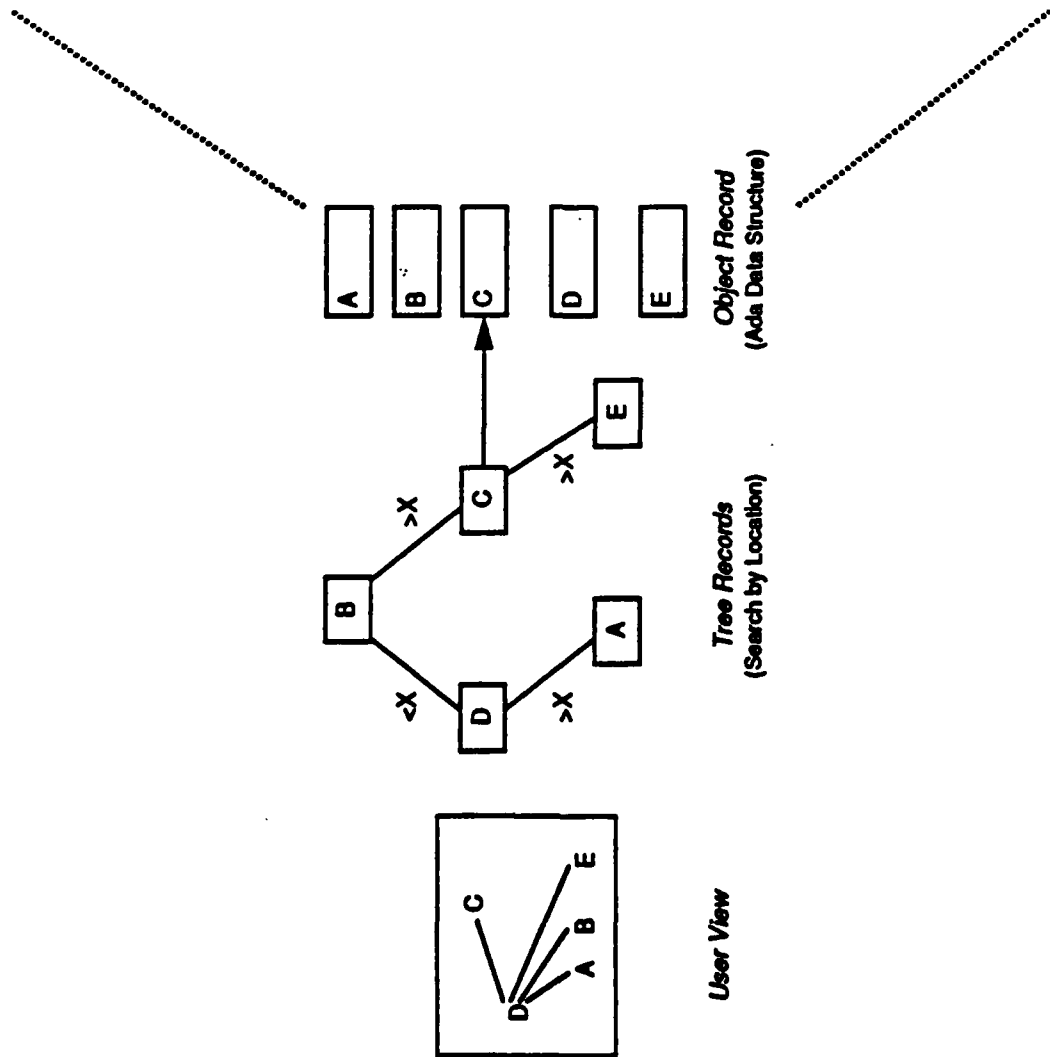
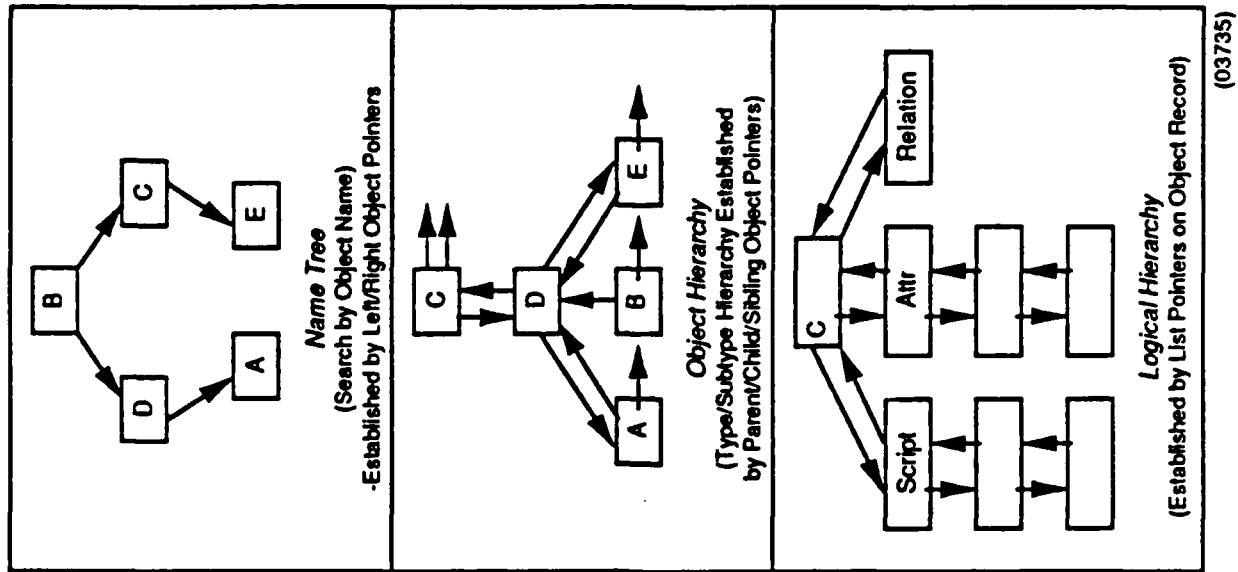


Figure 5-3, DSME Data Structure

SYSTEM		VIEW		GENEALOGY		OBJECT		SCREEN	
LOAD	L	CURSOR	CUR	CREATE OBJECT	C	RUN OBJECT SCRIPT	R_S	ATTRIBUTES	ATT
SAVE	S	REDISPLAY	RED	CREATE MANY OBJECTS	C_M	SHOW OBJECT	S_C	RELATIONSHIPS	REL
HELP	H	SCROLL	SCL	ADD LINK	A_L			SCRIPTS	SCR
EXIT	E	SHOW ALL	S_A	DELETE LINK	D_L				
		MOVE OBJECT	M	DELETE OBJECT	D				
		MOVE TREE	M_T						
COMMAND: <COMMAND>									
SYSTEM STATUS MESSAGES									

(03811)

Figure 5-4, DSME Genealogy Screen

Table 5-1, Genealogy Screen Command Definitions

Command	Abbreviation	Description	Section
Add Link	A_L	Adds parent/child relationship between 2 objects	5.1.3.3
Attributes	ATT	Moves to the Attributes screen	5.1.5.1
Create Many Objects	C_M	Create many objects similar objects under a parent	5.1.3.2
Create Object	C	Establishes an object in the genealogy	5.1.3.1
Cursor	CUR	Gives the user cursor control	5.1.2.1
Delete Link	D_L	Removes parent/child relationship	5.1.3.4
Delete Object	D	Removes object from genealogy	5.1.3.5
Exit	E	Exits User Interface	5.1.1.4
Help	H	Not available at this time	5.1.1.3
Load	L	Loads a genealogy from the database	5.1.1.1
Move Object	M	Relocates an object on the screen	5.1.2.5
Move Tree	M_T	Not available at this time	5.1.2.6
Redisplay	RED	Erases and redraws the screen	5.1.2.2
Relationships	REL	Moves to the Relationships screen	5.1.5.2
Run Object Script	R_S	Not available at this time	5.1.4.1
Save	S	Saves a genealogy structure to the database	5.1.1.2
Scripts	SCR	Moves to the Scripts screen	5.1.5.3
Scroll	SCL	Scrolls the screen	5.1.2.3
Show All	S_A	Not available at this time	5.1.2.4
Show Object	S_O	Not available at this time	5.1.4.2

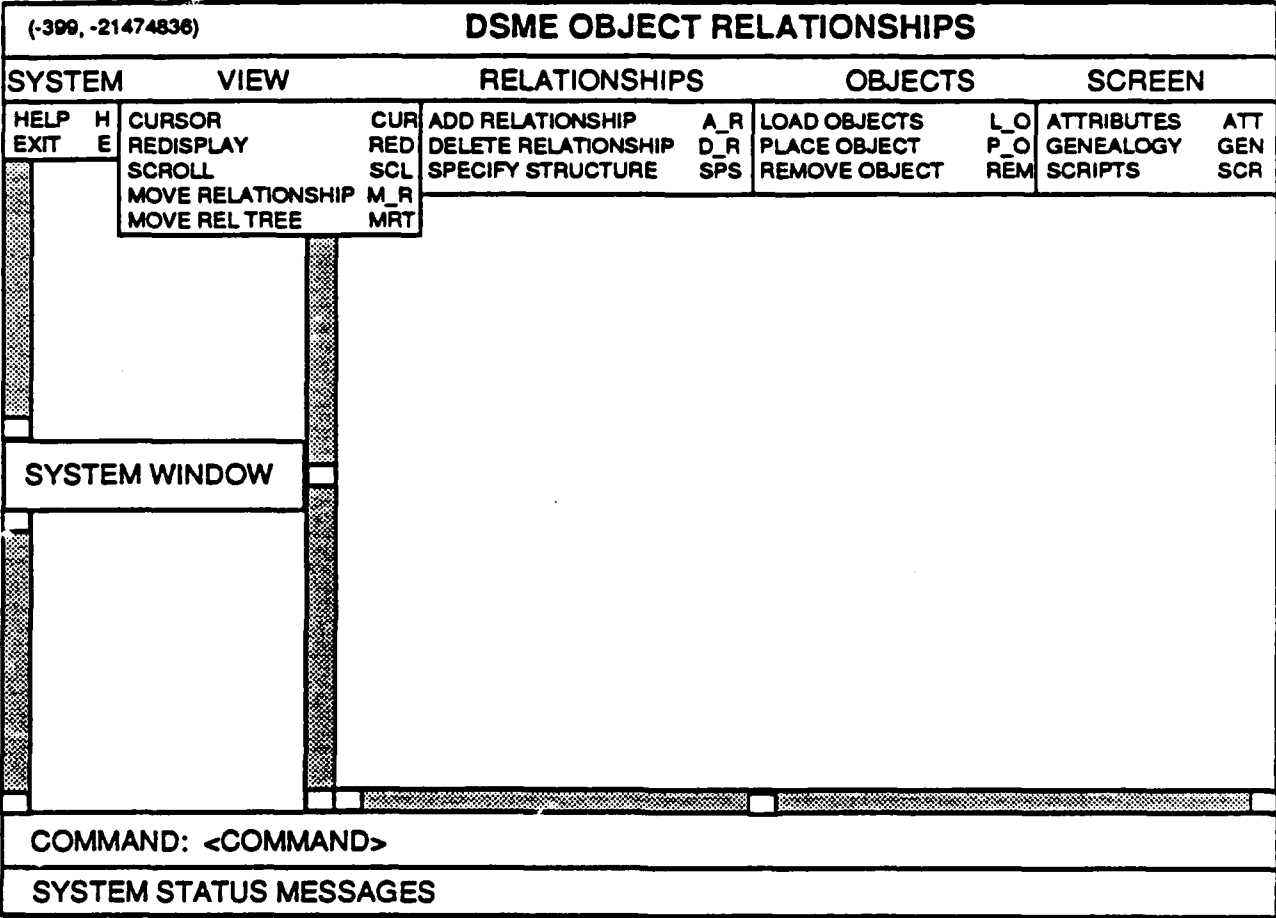
DSME OBJECT ATTRIBUTES									
SYSTEM		VIEW		ATTRIBUTE		OBJECT		SCREEN	
HELP	H	CURSOR	CUR	ADD ATTRIBUTE	A_A	LOAD OBJECTS	L_O	GENEALOGY	GEN
EXIT	E	REDISPLAY	RED	DELETE ATTRIBUTE	D_A	SELECT OBJECT	SEL	RELATIONSHIPS	REL
		INHERIT	I	EDITS ATTRIBUTE	E_A			SCRIPTS	SCR
		STAND ALONE	STA						
SYSTEM WINDOW									
COMMAND: <COMMAND>									
SYSTEM STATUS MESSAGES									

(03612)

Figure 5-5, DSME Attribute Screen

Table 5-2, Attributes Screen Command Definitions

Command	Abbreviation	Description	Section
Add Attribute	A_A	Adds an attribute to an object	5.1.8.1
Cursor	CUR	Gives the user cursor control	5.1.7.1
Delete Attribute	D_A	Removes an attribute from an object	5.1.8.2
Edit Attribute	E_A	Changes the value of an attribute	5.1.8.3
Exit	E	Exits User Interface	5.1.6.2
Genealogy	GEN	Moves to the Genealogy screen	5.1.10.1
Help	H	Not available at this time	5.1.6.1
Inherit	I	Not available at this time	5.1.7.3
Load Objects	L_O	Places a list of objects in window	5.1.9.1
Redisplay	RED	Erases and redraws screen	5.1.7.2
Relationships	REL	Moves to the Relationships screen	5.1.10.2
Scripts	SCR	Moves to the Scripts screen	5.1.10.3
Select Object	SEL	Selects an object for future commands	5.1.9.2
Stand Alone	STA	Not available at this time	5.1.7.4



(03613)

Figure 5-6, DSME Relationship Screen

Table 5-3, Relationships Screen Command Definitions

Command	Abbreviation	Description	Section
Add Relationship	A_R	Adds a relationship between two objects	5.1.13.1
Attributes	ATT	Moves to the Attributes screen	5.1.15.1
Cursor	CUR	Gives user cursor control	5.1.12.1
Delete Relationship	D_R	Deletes a relationship between two objects	5.1.13.2
Exit	E	Exits User Interface	5.1.11.2
Genealogy	GEN	Moves to the Genealogy screen	5.1.15.2
Help	H	Not available at this time	5.1.11.1
Load Objects	L_O	Places list of objects in window	5.1.14.1
Move Relation	M_R	Relocates an object on screen	5.1.12.4
Move Rel Tree	MRT	Not available at this time	5.1.12.5
Place Object	P_O	Places an object on the screen	5.1.14.2
Redisplay	RED	Erases and redraws screen	5.1.12.2
Remove Object	REM	Removes an object from the screen	5.1.14.4
Scripts	SCR	Moves to the Scripts screen	5.1.15.3
Scroll	SCL	Scrolls the screen	5.1.12.3
Specify Structure	SPS	Implements internal structure	5.1.13.3

DSME OBJECT SCRIPTS									
SYSTEM		OBJECTS		SCRIPTS		ACTIONS		SCREEN	
HELP	H	REDISPLAY	RED	ADD SCRIPT	ADS	CREATE ACTION	CRA	ATTRIBUTES	ATT
EXIT	E	LOAD OBJECTS	L_O	DELETE SCRIPT	DES	MODIFY ACTION	MOD	RELATIONSHIPS	REL
		SELECT OBJECT	SEL	EDIT SCRIPT	EDS	INSERT ACTION	INS	GENEALOGY	GEN
				SELECT SCRIPT	SES	DELETE ACTION	DEL		
				COPY SCRIPT	COS				
				RUN SCRIPT	RUS				
SYSTEM WINDOW									
COMMAND: <COMMAND>									
SYSTEM STATUS MESSAGES									

(03614)

Figure 5-7, DSME Script Screen

Table 5-4, Scripts Screen Command Definitions

Command	Abbreviation	Description	Section
Add Script	ADS	Adds a script to an object	5.1.18.1
Attributes	ATT	Moves to the Attributes screen	5.1.20.1
Copy Script	COS	Copies a script from another object	5.1.18.5
Create Action	CRA	Creates a new action for an object	5.1.19.1
Delete Action	DEL	Deletes an action from a script	5.1.19.4
Delete Script	DES	Deletes a script from an object	5.1.18.2
Edit Script	EDS	Selects a script for action commands	5.1.18.3
Exit	E	Exits User Interface	5.1.16.1
Genealogy	GEN	Moves to the Genealogy screen	5.1.20.3
Help	H	Not available at this time	5.1.16.1
Insert Action	CRA	Inserts an action before another action	5.1.19.3
Load Objects	L_O	Places a list of objects classes in window	5.1.17.2
Modify Action	MOD	Changes an action	5.1.19.2
Redisplay	RED	Erases and redraws screen	5.1.17.1
Relationships	REL	Moves to the Relationships screen	5.1.20.2
Run Script	RUS	Not available at this time	5.1.18.6
Select Object	SEL	Selects an object for further commands	5.1.17.3
Select Script	SES	Changes a scripts condition	5.1.18.4

These packages are described in the following paragraphs.

Package - DATABASE_INTERFACE

This package contains the procedures necessary to access the RIM database. Each procedure corresponds to a FORTRAN subroutine. This package also retrieves and stores data items within the RIM database. The method of accessing the database consists of using Ada routines to format the data in a manner required by the RIM database package. These routines then call Ada routines that are mapped to FORTRAN routines which call the appropriate RIM commands.

This method of access to the RIM database was chosen in order to limit the use of VAX Run Time Library routines (system-dependent routines). RIM passes different data types back in a single integer array which then would have needed to be decomposed into the different types. Ada does not support this activity. Therefore, VAX Run Time Library routines would have had to be used, which would have increased the number of system-dependent routines and made the system less portable.

Package - OBJECTS

An object is any physical entity, or class of entities, in the system being modeled. This package contains the data definition of an object. Each object includes an attribute list, relation list, and a script list that are managed by packages ATTRIBUTES and SCRIPTS. This package also performs all operations on the object record. It establishes the genealogy of objects (parent/child relationships), assigns the object name and kind, and keeps track of whether the object needs to be saved to the database. In addition to these operations, the Objects package also drives the Attributes, Scripts, and Actions packages for assignment of attributes, relations, scripts, and actions for an object. The package Objects drives other packages by performing the selected operation at a high level (getting the various inputs needed from the user and performing error checking) and then calling other routines in other packages for the more detailed operations (updating attribute, relation, script, or action records).

Package - ATTRIBUTES

An attribute is a property of an object. Structural relationships are also contained in this package. This package contains the data definition of an attribute, a relationship, and procedures that can influence them. This package also performs all of the operations on the attribute record and relation record. It creates and maintains the linked lists for attributes and relations. In addition, for the attribute record, it assigns the attribute value, attribute type, and attribute name. It also keeps track of whether or not the attribute record needs to be saved to the database. For the relation record, it assigns the relation name, relation type, attribute name (if the relation type is a binder), the location of the relation (object) on the relationship screen, and the pointer to the related object's relation record; it also keeps track of whether the relation record needs to be saved to the database.

Package - SCRIPTS

This package contains the necessary functions and procedures to implement the scripts data structure. Script records are stored in a linked list. The script record contains the name of the script, a condition that can be evaluated to determine whether or not to execute some actions, and a pointer to a linked list of actions. This package also performs all the operations on the script record and action record. It adds actions to a given script's list of actions, and adds scripts to an object's script list.

Package - LOCATION_TREES

Location trees keep track of where an object should be displayed on the screen. Two trees keep track of the location of each object that is a part of the genealogy screen and one tree keeps track of the location of each object that is a part of the relationship screen. This package also performs all the operations on the tree record. It stores and removes object names, locations, and a pointer to the object in the object tree. It also modifies the location information of an object. In addition to these operations, the Location Trees package drives any high level-object and database operations that use the location trees. These high-level routines then call the appropriate lower-level routines from the appropriate package.

Package - SCR_CONST

This package consists of all the constants that refer to screen coordinates for the DSME User Interface system. These screen coordinates are in ReGIS graphics units because ReGIS graphics functions are used to allow the user to select locations on the screen. Horizontal values are referred to as 'X' values and vertical values are referred to as 'Y' values. The ReGIS coordinate system is defined as follows: the upper left corner of the video screen has coordinates: X=0 and Y=0; and the lower right corner of the video screen has coordinates: X=799 and Y=479.

Package - GRAPHICS

This package contains all the procedures and functions necessary to control graphics and text on the screen. This package also manipulates the video display of the user's terminal. The graphics routines utilize SMG routines to display screen borders, menus, overlay boxes, and windows. ReGIS graphics routines are used for the objects in the work area as well as the links drawn between them, and for the cursor.

Package - TOOLS

This package contains tools that the DSME User Interface routines use, such as string manipulation routines. It also contains common structures used by the DSME User Interface routines (such as linked lists). This package performs operations on the node record. It creates new names, performs various operations on strings, and converts various data types to strings or reals. Also, the Tools package provides conversion from virtual

graphics coordinates to ReGIS graphics coordinates. The virtual graphics coordinates are established in the System Specific Tasks unit and are capable of ranging from the smallest number a system can store to the largest number a system can store. ReGIS only recognizes coordinates in a specific range. Therefore, procedures like Convert Graphics To Virtual and Convert Virtual To Graphics are needed for ReGIS to display objects on the screen in the proper locations.

Package - PARSER

The Parser package recursively examines an expression for valid syntax. An expression is evaluated at the time it is entered by the user. The parser operates under the following grammar:

EXPRESSION ::=

SUBEXPRESSION RELATIONAL_OPERATOR {CONSTANT |
SUBEXPRESSION}

SUBEXPRESSION ::=

{VALUE OPERATOR VALUE |
VALUE OPERATOR SUBEXPRESSION OPERATOR VALUE |
VALUE}

RELATIONAL_OPERATOR ::=

{ < | > | >= | <= | = | /= }

OPERATOR ::=

{ + | - | * | / }

VALUE ::=

T {CONSTANT} |
B {CONSTANT} |
D {CONSTANT} |
OBJECT NAME, ATTRIBUTE NAME

CONSTANT ::=

{integer}

Each token in the expression must be delimited by one space character with the exception of those VALUES that consist of a character immediately followed by an integer.

Package - ACTIONS

An action is a task that is evaluated for execution of an activity. This package contains the data definition and the routines to manipulate a linked list of actions. This package also performs operations on the action record. It inserts new actions, modifies

actions, and deletes actions from an action list.

Package – SYSTEM_SPECIFIC_TASKS

This package performs system-dependent operations. It provides an interface to various VAX run-time library routines that perform file manipulations and spawning. It also establishes the virtual screen limits for the system and performs operations that use these limits.

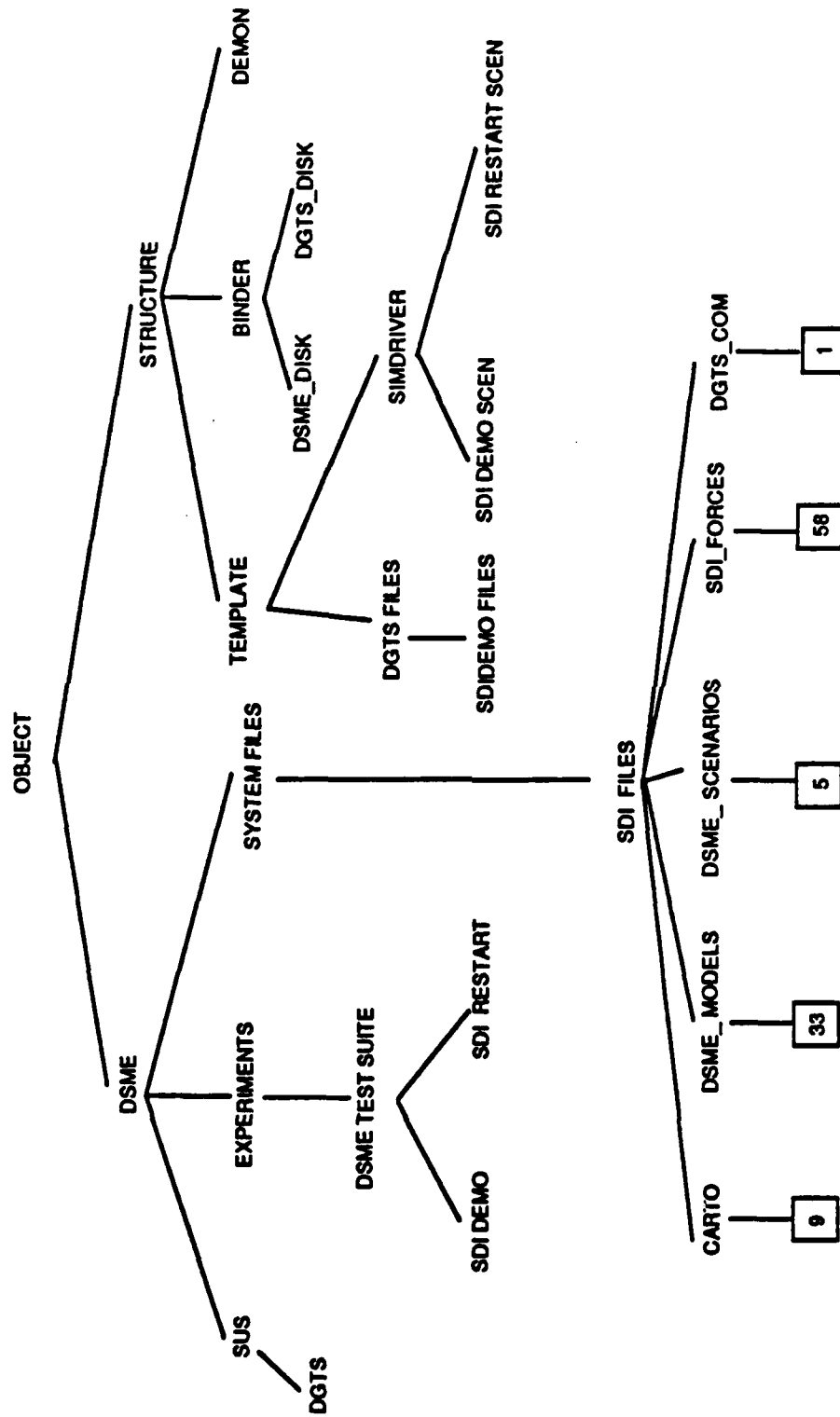
5.4 DSME DATABASE

The User Interface software was used to create a database with information describing the SUS (Simulation Driver Integration) to support the execution of experiments using SIM DRIVER. This section documents the high-level database organization, specifically the Genealogy and Relationship structures that were created for the DSME Concept Demonstration system.

Figures 5-8 through 5-11 show examples of the contents of the DSME Demonstration database created under the DSME effort. The DSME Demonstration database contains the definition of the Simulation Driver Integration system.

Figure 5-8 shows the Genealogy structure of the Simulation Driver Integration system. Five objects make up the skeleton structure of the Genealogy Screen. These objects are: "OBJECT," "STRUCTURE," "TEMPLATE," "BINDER," and "DEMON." These objects are the basic structure upon which System Under Study (SUS) definitions must build. Beneath the object named "DSME" are the components necessary to define a System Under Study: "SUS," "EXPERIMENTS," and "SYSTEM FILES." Beneath the object name "SUS" are the Systems Under Study that have been defined in the database (i.e., DGTS). All of the defined experiments are beneath the object "EXPERIMENTS." An experiment allows the user to execute a SUS in different ways, in order to compare the results of modifying the way in which the SUS is run. In this case, the "DSME TEST SUITE" of experiments includes: the "SDI DEMO" experiment and the "SDI RESTART" experiment. All the files that may be used to execute the SUS are under the object "SYSTEM FILES." The "SDI FILES" object contains all of the data files to run the Simulation Driver Integration system. Five categories of files are needed for the Simulation Driver Integration system: 1) cartographic data files (9 files beneath the object named "CARTO"); 2) Simulation Driver Integration model files (33 files beneath the object named "DSME_MODELS"); 3) initial order and edf command files (5 files under the "DSME_SCENARIOS" object); 4) military force data files (58 files beneath the "SDI_FORCES" object); and 5) DGTS logical name definition files (1 file beneath the "DGTS_COM" object).

The three different types of object structures that may be defined are beneath the object name "STRUCTURE." These structures allow the user to create relationships other than the parent/child relationships defined on the Genealogy screen. A "TEMPLATE" structure is used to establish a set of objects with similar relationships but different object



(03595)

Figure 5-8, Simulation Driver Integration Genealogy Structure

instances that belong to each relationship. A "BINDER" structure is used to establish a group of objects that inherit attribute values of non-parent objects. A "DEMON" structure is used to initiate scripts upon entering a given system state. No demon structures were defined for the DSME Demonstration database since the execution of scripts within the User Interface has not been yet implemented.

The objects that use the template structure on the Relationship screen are below the "TEMPLATE" object. For the DSME Demonstration database these objects are: the "DGTS FILES" and the "SIMDRIVER" objects. The template relationships of these objects are shown in Figures 5-10 and 5-11 and will be described when those figures are discussed. The object "SDIDEMO FILES," since it is a child of the "DGTS FILES" object, is an instance of that object, meaning it will follow the same template structure as defined for the "DGTS FILES" object on the Relationship screen (see Figure 5-10).

Similarly, both the "SDI DEMO SCEN" and "SDI RESTART SCEN" objects are children of the "SIMDRIVER" object; thus, they follow the template structure defined for the "SIMDRIVER" object on the Relationship screen (see Figure 5-11).

Beneath the "BINDER" object are the objects that use the binder structure on the Relationship screen. For the DSME Demonstration database these objects are the "DSME_DISK" and "DGTS_DISK" objects. The Binder relationships are defined for these objects on the Relationship screen and are described under the discussion of Figure 5-10.

Figure 5-9 contains an example of attributes that may be defined for an object on the Attributes Screen of the User Interface. This figure contains the attributes of the object "SDI DEMO SCEN." In this example, the attributes define how the experiment "SDI DEMO SCEN" will be executed. For Simulation Driver Integration experiments, these attributes pertain to DGTS qualifiers or options available when generating a scenario. A value of "TRUE" means that the particular DGTS qualifier will be enabled for that experiment. The "EDF FILE OBJECT" attribute specifies the name of the object that has the relationships pointing to each of the 42 data files needed to run a Simulation Driver Integration experiment. The attributes "SPEC1" and "SPEC1B" are created when the relationships on the Relationship screen have been established. The "SPEC1" attribute describes the specification that must be satisfied in order for "SPEC1B" to be "TRUE." In Figure 5-9 the value " $T1 + T2 + T3 = 4$ " for the attribute "SPEC1" means that for the relationships labeled "T1," "T2," and "T3" on the Relationships screen, there should be a total of four children.

Figures 5-10 and 5-11 show the relationships established on the Relationships screen for the "BINDER" and "TEMPLATE" objects (i.e., "DGTS_DISK," "DSME_DISK," "DGTS FILES," "SDIDEMO FILES," "SIMDRIVER," "SDI DEMO SCEN," and "SDI RESTART SCEN").

OBJECT: SDI DEMO SCEN

DESCRIPTOR:	SIM DRIVER DEMONSTRATION
PURPOSE:	DEMONSTRATE MODEL CAPABILITY
MODEL ADDRESS LISTING:	FALSE
SCENARIO ARCHIVE LISTING:	FALSE
REAL TIME CLOCK CONTROL:	FALSE
DURATION LIMIT CONTROL:	FALSE
EDF COMMAND FILE:	TRUE
EDF FILE OBJECT:	SDIDEMO FILES
EVENT TRACE LISTING:	FALSE
SCENARIO ORDER LISTING:	FALSE
MESSAGE TRACE LISTING:	TRUE
LOCAL MONITOR DISPLAY:	FALSE
RESTART SCENARIO:	FALSE
RESTART FILE NAME:	NULL
RESTART FILE DIRECTORY:	NULL
PERF SUMMARY LISTING:	FALSE
UPDATE INTERVAL CONTROL:	FALSE
WARNING MESSAGE SUPPRESS:	FALSE
SPEC1:	$T1 + T2 + T3 = 4$
SPEC1B:	TRUE

(03769)

Figure 5-9, Simulation Driver Integration Attribute Example

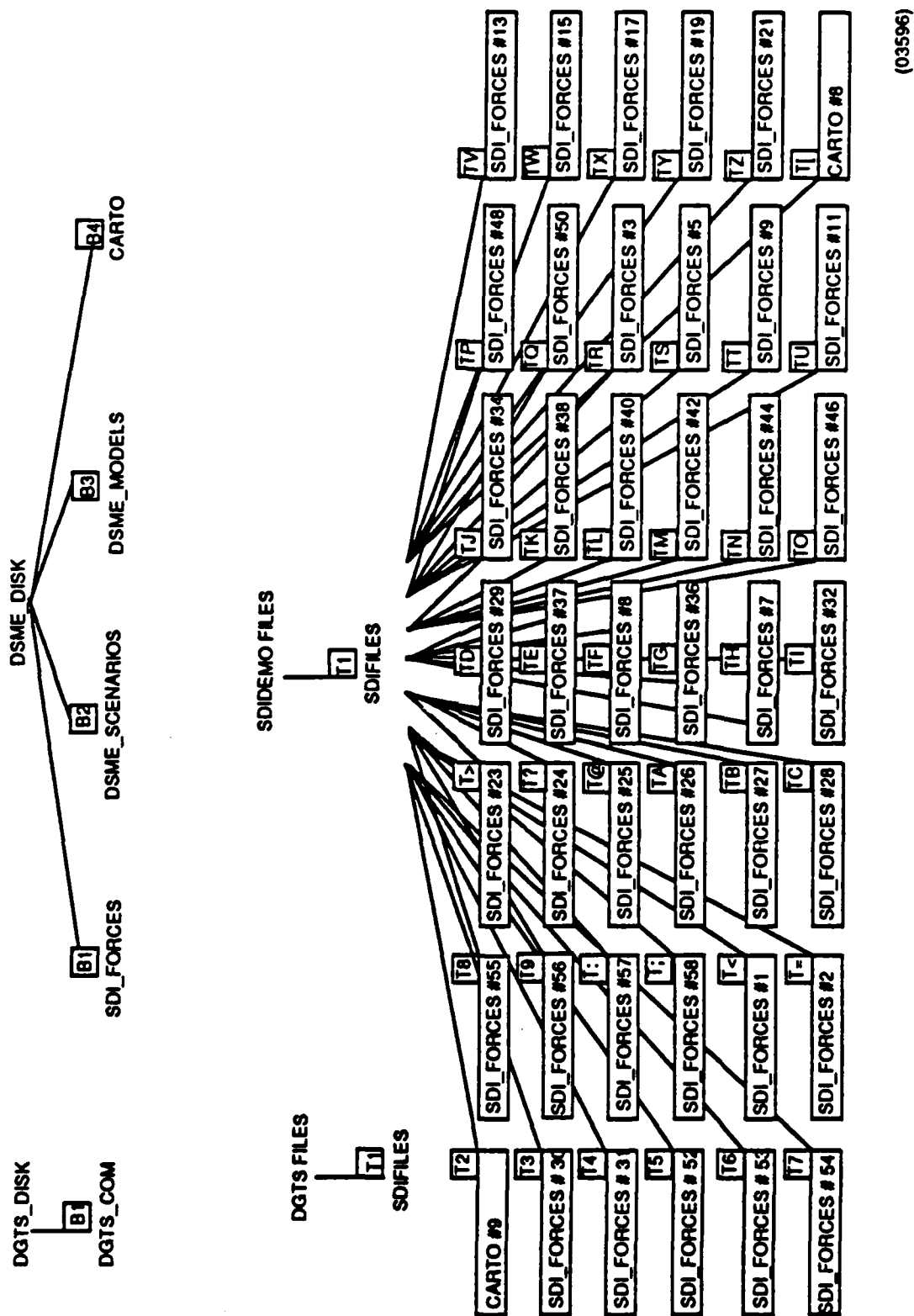
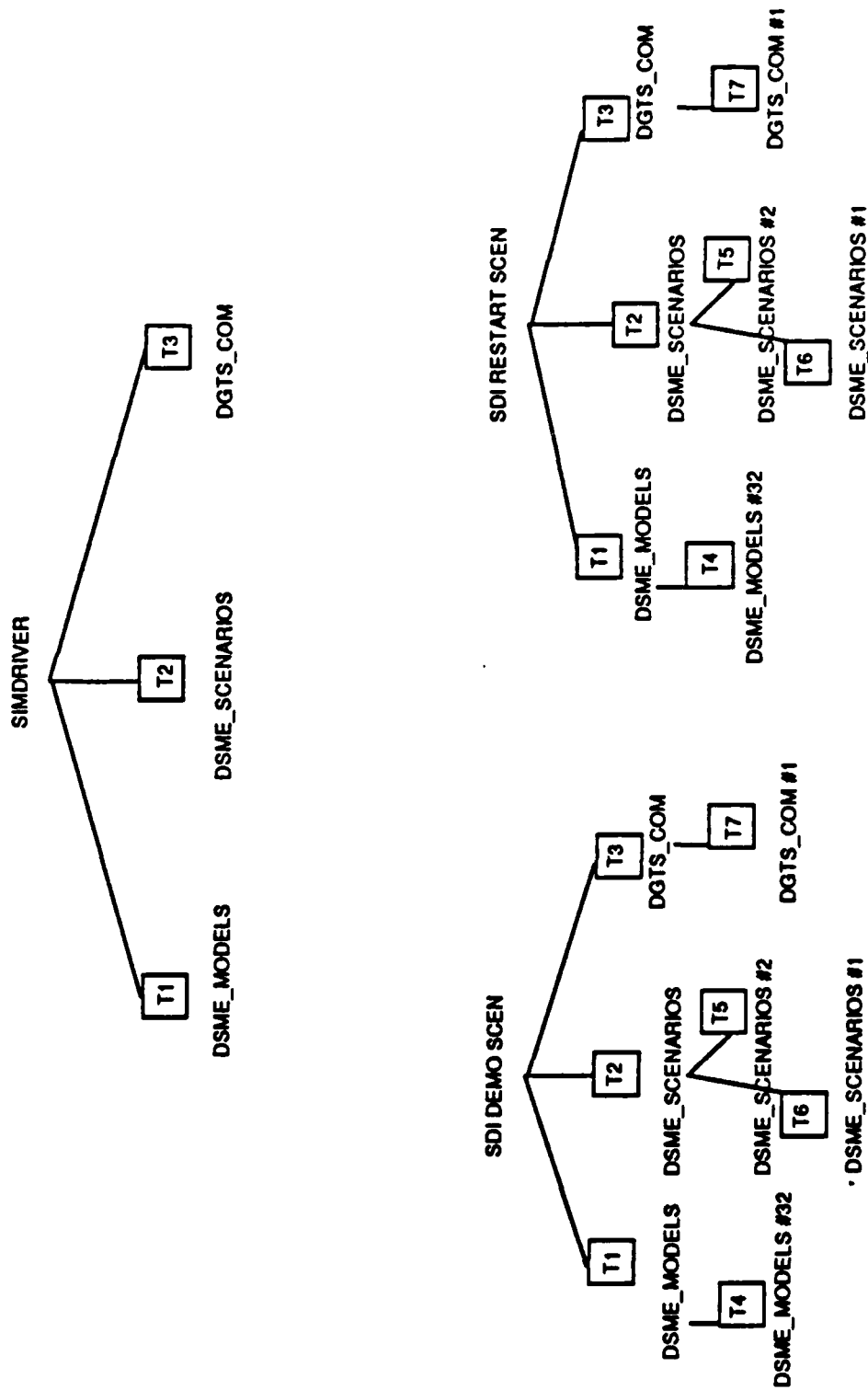


Figure 5-10, Simulation Driver Integration Relationship Structure

(03596)



(03594)

Figure 5-11, Simulation Driver Integration Relationship Structure (Cont'd)

The Binder Relationship defined for the objects "DGTS_DISK" and "DGTS_COM" specifies that the value of the common attribute between the two objects is determined by the "DGTS_DISK" (i.e., the parent) object. That is, the value of a particular attribute of the "DGTS_COM" object is bound to the same attribute of the "DGTS_DISK" object. Simply put, their values will always be the same.

Similarly, the Binder Relationship defined for the objects "DSME_DISK" and "SDI_FORCES," "DSME_SCENARIOS," "DSME_MODELS," and "CARTO" specifies that the value of the common attribute between all the objects is determined by the "DSME_DISK" (i.e., the parent) object.

The template relationship defined for the "DGTS FILES" and "SDIFILES" is labeled "T1." The structure specified for this template is $T1 = 42$. Although this specification cannot be seen on the Relationship screen, it may be viewed as the "SPEC1" attribute on the Attributes screen for the "DGTS FILES" object. In this case the specification of $T1 = 42$ means that 42 files are necessary for a DGTS experiment.

The "SDIDEMO FILES" object is an instance of the "DGTS FILES" template, because it is a child of "DGTS FILES," which is a template structure. Therefore, it has the same requirement of $T1 = 42$. As shown in Figure 5-10, 42 objects are linked to the "T1" relationship and those 42 links satisfy the requirement $T1 = 42$. The 42 objects represent the particular data files to be used to execute the Simulation Driver Integration experiments. The user can find out the name of the file, the type of file, and other information for each object by looking at the attributes defined for the object on the Attributes screen.

The template relationships defined for the "SIMDRIVER," "DSME_MODELS," "DSME_SCENARIOS," and "DGTS_COM" are labeled "T1," "T2," and "T3." The structure specified for this template is $T1 + T2 + T3 = 4$. Although this specification cannot be seen on the Relationship screen, it may be viewed as the "SPEC1" attribute on the Attributes screen for the "SIMDRIVER" object (an example is in Figure 5-9). In this case the specification of $T1 + T2 + T3 = 4$ means that four files are necessary for a DGTS experiment.

The "SDI DEMO SCEN" object is an instance of the "SIMDRIVER" template, because it is a child of "SIMDRIVER," which is a template structure. Therefore, it has the same requirement of $T1 + T2 + T3 = 4$. As shown in Figure 5-11, four objects are linked to the "T1," "T2," and "T3" relationships and those four links satisfy the requirement $T1 + T2 + T3 = 4$. The four objects represent the particular initial order file and command files that are to be used to execute the "SDI DEMO SCEN" experiment. The user can find out the name of the file, the type of file, and other information for each object by looking at the attributes defined for the object on the Attributes screen.

The "SDI RESTART SCEN" object is also an instance of the "SIMDRIVER" template, because it is a child of "SIMDRIVER," which is a template structure. Similarly, it has the same requirement of $T1 + T2 + T3 = 4$. As shown in Figure 5-11, four objects are

linked to the "T1," "T2," and "T3" relationships and those four links satisfy the requirement $T1 + T2 + T3 = 4$. The four objects represent the particular initial order file and command files that are to be used to execute the "SDI RESTART SCEN" experiment. The user can find out the name of the file, the type of file, and other information for each object by looking at the attributes defined for the object on the Attributes screen.

5.5 EXPERIMENT PREPARATION COMPONENT

As previously indicated, the Experiment Preparation Component creates the command files necessary to execute the Simulation Driver Integration software by extracting the necessary information from the DSME database. This section describes in more detail how the Experiment Preparation component was implemented.

This software component is relatively simple. Two sets of procedures were written. One set simply provides an interface from the RIM database access procedures, which are written in FORTRAN, to the Experiment Preparation component, which is written in Ada. The other set of procedures provides for the access of the necessary data from the database and construction of the command files. When the Experiment Preparation component is executed, the user provides the name of the DSME database file. Via interactive prompts and responses, the user provides the Experiment Preparation component with the name of the Experiment to be executed. The Experiment Preparation component then accesses the DSME database and locates all the DGTS information needed to run the selected experiment. The required information includes all the necessary data files, the model to be used (in this case the Simulation Driver Integration model), and all necessary command files. Once all items have been located and verified, the command file to run the experiment is created.

5.6 SYSTEM UNDER STUDY: SIMULATION DRIVER INTEGRATION

The System Under Study in the Concept Demonstration system is the Simulation Driver Integration simulation capability. This section briefly describes the Simulation Driver Integration system, and then describes the modifications that were required to SIM DRIVER in order for it to be integrated into the DSME Concept Demonstration system.

5.6.1 *Simulation Driver Integration System Description*

The Simulation Driver Integration project combined the Dynamic Ground Target Simulator (DGTS), a general-purpose battlefield scenario generation system, with the Tactical Air Surveillance Radar Netting (TASRAN) simulation model, which simulates several types of ground-based and airborne air surveillance radars (TPS-43, Patriot, E3-A, AASR) to provide a combined air-ground battlefield simulation capability. The primary challenge of this effort was the integration of two very different simulation systems. The result was a distributed simulation which runs on as many as four VAX processors connected by DECnet communication links. It is currently installed in the RADC Command and Control Laboratory. This section provides an overview of the SIM DRIVER, as well as the overall architecture of the resultant product. A more detailed

description of the project is provided in Appendix A of the *Functional Description*.

The objectives of the Simulation Driver Integration project were:

1. to define the requirements of scenario generation tools which integrate the existing capabilities of the Dynamic Ground Target Simulator (DGTS) scenario generation system and associated battlefield simulation model, with the Tactical Air Surveillance Radar Netting (TASRAN) simulation model and the Tactical Communications (TACOM II) simulation model, to provide a capability to generate realistic air/ground battlefield scenarios;
2. to develop the capability to generate realistic air/ground battlefield scenarios using the assets of the DGTS system, the BSG model, the TASRAN model, and the TACOM II model; and
3. to demonstrate the utility of the scenarios for evaluating air/ground battle management techniques within the context of the RADC C³I Laboratory Complex.

5.6.1.1 DGTS Scenario Generation System

The Dynamic Ground Target Simulator (DGTS) is a software-based system for developing and generating detailed battlefield scenarios to support the development and testing of battlefield management systems. Its purpose is to provide a means for generating message streams and/or files describing the detailed behavior of military units and their associated equipment, along with the environment within which such units operate. This information can be used as input to a battlefield management system responsible for interpreting data collected from a battlefield situation and reacting to that situation. The DGTS system operates on VAX computers running the VAX/VMS operating system. Most of the DGTS software is implemented in VAX-11 Pascal with a few key procedures implemented in VAX-11 Macro assembler language for greater speed and efficiency.

5.6.1.2 Battlefield Simulation Model

This section describes the structure and capabilities of the DGTS battlefield simulation model that was the baseline for the Simulation Driver Integration program. The capabilities of this model are described in terms of the types of entities which are represented, the types of events in which the entities can be involved, and the types of messages that are output to various subscriber processes.

Entities are aspects of, or objects within, the real world which are to be represented in scenarios. The following types of entities are defined in this model:

- military units (ranging in size from platoons to corps/armies);
- platforms (aircraft, vehicles, and fixed sites);
- equipment (radars, radios, jammers, sensors, and weapons);

- ground and air formations, which control the movement of hierarchical groups of vehicles and aircraft;
- command and control nodes (commanders, staff officers, etc.); and
- communications networks.

Military units are defined hierarchically. High-level units (battalions and larger) are made up of smaller units. Low-level units (companies and platoons) are made up of individual vehicles, aircraft, or fixed sites, which are collectively referred to as platforms. Various types of equipment, including radios, radars, jammers, sensors, and weapons, can be mounted on each platform. Each platform also carries a command and control node of a particular type, which controls the actions of the platform and its equipment in response to scenario events.

5.6.1.3 Tactical Air Surveillance Radar Netting Model

The Tactical Air Surveillance Radar Netting (TASRAN) simulation is a medium-to-high fidelity simulation for evaluating netted tactical air surveillance systems. It simulates arbitrary types and numbers of ground-based and airborne surveillance and tracking radars, air surveillance radar targets (aircraft, cruise missiles, helicopters), communication links, and operations centers. Air-to-air engagement is also modeled. TASRAN was designed to provide detailed technical insight into scenarios including reasonably large numbers of radars, communication links, and attacking aircraft.

Four classes of radars are modeled in the TASRAN simulation: 1) conventional rotating surveillance radars operating in a track-while-scan mode, such as the AN/TPS-43E, 2) stationary phased-array radars operating under computer control, such as Patriot, 3) airborne track-while-scan radars (E3-A), and 4) airborne conformal phased-array radars (AASR).

The simulation operates at a measurement-by-measurement level. Radar measurements are processed into tracks in the radar data processor module, which includes algorithms for track initiation, association, update, and track file maintenance. Anti-jamming and ESM modes are also modeled.

Local tracks generated by the individual radars are all sent to an Operations Center, where the local tracks are combined into system tracks. The quality of these system tracks defines the worth of the netted radar system. The system tracks are also used by the air-to-air engagement part of the simulation.

A typical TASRAN simulation scenario consists of a number of raids of hostile aircraft (bombers and fighter cover) heading toward friendly targets. These hostile aircraft may carry ECM and other emitters. Stand-off jammers may also be included. The netted radar system performs surveillance and tracking of the raids, producing system tracks that are used by the engagement model. Interceptor aircraft are assigned and committed to engagements from one or more air bases. The interceptors are vectored to the incoming

raid(s), where they acquire the raid and perform the air-to-air engagement.

5.6.1.4 Simulation Driver Integration Architecture

The structure of the Simulation Driver Integration software that has resulted from this approach is shown in Figure 5-12. An experiment using the Simulation Driver Integration software may involve up to four separate, concurrently executing processes, as described below:

- an enhanced version of the DGTS Scenario Executive process, controlling the execution of an enhanced version of the Blue Scenario Generator battlefield simulation model (called the Simulation Driver Integration model);
- an enhanced version of the DGTS Scenario Monitor process, supporting the dynamic display and interactive manipulation of the scenarios as they are generated;
- the TASRAN Subscriber Process, derived from the TASRAN/TACOM software, performing a detailed simulation of the detection, tracking, and communications associated with the Blue air surveillance radar network; and
- the TASRAN Monitor process, derived from the DGTS Scenario Monitor, providing dynamic interactive displays of the air threat as perceived by the Blue air surveillance radar network simulated by TASRAN.

These processes can be configured on one or more VAX processors within the RADC C³I Laboratory Complex in a variety of ways, ranging from running all four processes on a single VAX to running each process on a separate VAX in a separate facility. The processes communicate with one another by asynchronously passing messages through full-duplex channels provided by DECnet.

The DGTS Scenario Executive controls the execution of the Simulation Driver Integration model to produce an integrated air-ground battlefield scenario. Aircraft positions and velocities, along with relevant environmental information, such as jammer emissions and chaff cloud descriptions, are passed to the TASRAN Subscriber Process, which simulates the acquisition and tracking functions of the air surveillance network. The TASRAN Subscriber Process also controls Blue interceptor assignments by sending scenario orders that assign and direct Blue interceptors back to the Simulation Driver Integration model. The TASRAN Monitor provides a dynamic color graphic representation of the perceived air situation from various points of view within the simulated Blue air surveillance network.

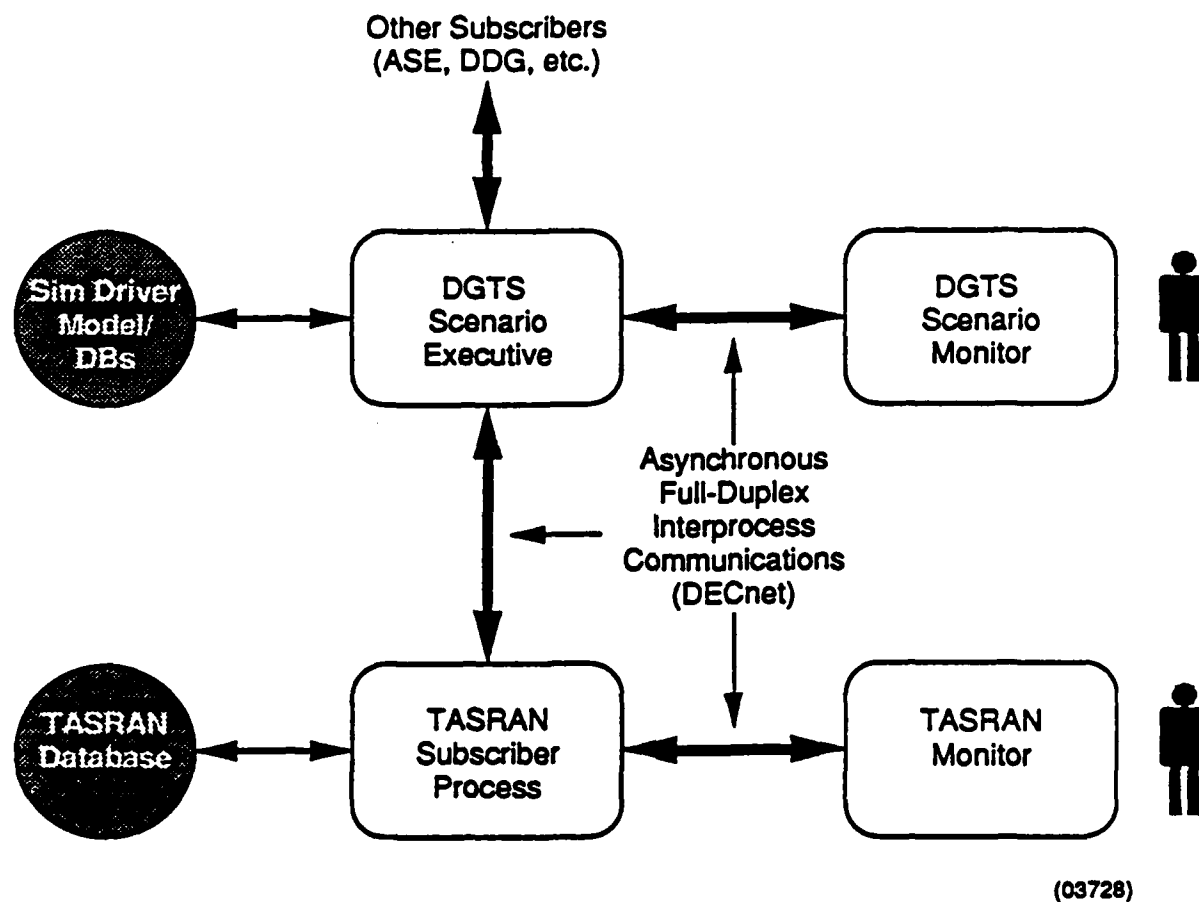


Figure 5-12, Simulation Driver Integration Software Architecture

5.6.2 *Modifications to Simulation Driver Integration*

In order to integrate the Simulation Driver Integration system into the DSME Concept Demonstration system, two categories of changes were required. One category included changes that supported the data collection control mechanisms which:

1. allow the user to specify which probes to turn on during which portions of the simulation, and
2. build the schedule for controlling when data is captured and stored.

The second category of modifications dealt with the specifics of the data collection itself, and included:

1. opening and closing the "Dump" files,
2. extracting data from global memory and converting it to the appropriate format, and
3. formatting the output messages.

These modifications are described in more detail in the following paragraphs.

The collection control modifications are not particularly unique to the Simulation Driver Integration software; that is, similar types of procedures would be required of any system being integrated into a DSME. These procedures performed two functions, as listed above. The first function is to establish, via an interactive dialog with the user, the data to be collected for a particular experiment, and the simulation time at which that data is to be collected. The primary reason for allowing the user to input this information is that there is a potentially very large set of data to be collected. Without some control of what data is collected for a particular experiment execution, a great deal of storage space would be wasted. The software that accomplishes this function is simple. The probes are coded based on the message facility of DGTS. For each probe, the user provides the information about if and when the data should be collected.

Based on this information, another procedure added to the SIM DRIVER Executive component then builds a linked list of probe "on/off" switches that is subsequently accessed by a Scheduling procedure.

The data collection procedures themselves that were added into the Sim Driver software include two basic types. One type is to retrieve a particular data item from global memory. There is a procedure for each of the following data types: string, real, integer, and enumerated type. The other type of procedure is the actual probe itself - that is, the procedure that directs the capture of particular items of data. Probe types implemented in the Concept Demonstration system include:

- Unit State,
- Platform State,
- Electronic Equipment State,
- C³I Action,
- Object Transition,
- Simulation Event State, and
- Sensor Report.

5.7 LOADER COMPONENT

The Loader component of the DSME Concept Demonstration system provides a bridge from the text (ASCII) output stream generated by DGTS and the input formats of the RIM DBMS. Specifically, the Loader reads individual messages stored in the DSME Dump file, which is populated with DGTS messages output as the result of execution of the probes. The appropriate data is extracted from the message, and inserted into a RIM command to store the information into the Performance Database maintained by RIM. There is no manipulation of the data elements themselves, as the Loader performs a strictly formatting task.

5.8 REPORTER COMPONENT

The Reporter component extracts performance data from the Performance Database and generates formatted tables containing the information. Here again there is no data manipulation. The Reporter consists of a set of predefined RIM queries. The description of the data to be extracted and the format of the output table are within the definition.

It should be noted that the Reporter component is merely a place holder for what would be a much more sophisticated statistical analysis and data presentation capability in the baseline DSME system.

6. LESSONS LEARNED AND FUTURE DIRECTIONS

The preceding sections document the results of the effort in terms of the concepts that evolved for a DSME and the software that was actually written in the Concept Demonstration system. This concluding section offers reflections on the lessons learned as a result of the effort, and the implications and recommendations for future directions.

In some respects, this is the most important section of the entire report. While some significant progress was made in the concept development and design, and while the Concept Demonstration system provided some insight into how an ultimate system might operate, this effort evolved primarily into an exercise in concept development, and many of the thoughts are not easily captured in either design or implementation products.

As the title of this section implies, this discussion is divided into two sections: 1) a summary of the lessons learned and observations made at the conclusion of the effort; and 2) some thoughts on potential next steps to further develop these concepts.

6.1 LESSONS LEARNED

A number of observations and lessons learned are documented in this section, specifically:

1. The design concepts and the data-flow charts are a useful building block for subsequent development.
2. The object-oriented user interface provides a useful baseline for database manipulation.
3. Much of distributed system performance analysis requires collection of the same data as non-distributed systems analysis requires.
4. The generality of the data collection mechanism remains a difficult issue. The probe concept was adequate in the Concept Demonstration system, but as implemented, is not a solution that can be generalized.
5. Commercial DBMS and statistical packages are an appropriate part of a DSME environment.

These topics are elaborated on in the following paragraphs.

Baseline Data Concepts/Data Flow

The first observation is simply that the concepts developed in the early part of the effort, and for that matter the top-level design represented in the data-flow diagrams of Section 4.1, provide a useful baseline for subsequent development. While this may seem a trivial point, it is an important one to note. Since the design was not completely implemented, and therefore not "validated," this observation could easily be dismissed; such

a dismissal would result in less than optimal utilization of the results of this effort.

Perhaps the most significant concepts that were developed, in terms of the overall conceptual development, were: 1) the overall perspective of the sequence of activities in distributed system development (analysis, simulation, prototype, operational), and 2) the notion of an environment populated with integrated tools that support all four aspects of the development process. As this concept evolved, it became apparent that most of the data collection and analysis functions required to support distributed systems analysis were common among all aspects of the development process; therefore, an integrated set of tools could provide several advantages, namely:

- Commonality, by using the same user interface, data collection, and data analysis tools regardless of whether the data is being collected from a simulation of a distributed system, a prototype of a distributed system, or the distributed system itself.
- Integrated data interpretation, by permitting the comparison of data from a simulation, a prototype, and an operational system. The simulation can be validated based on operational system results, and/or the operational system can be tuned based on simulation results.
- Efficiency, since one set of tools can support multiple aspects of the development process.

The breadth of applicability is also reflected in the data-flow diagrams themselves. At the top level, there is little to actually distinguish the functions as applying to specific aspects of distributed system development. Instead, they reflect an overall flow of data when defining, executing, and analyzing an experiment, with some emphasis placed on the unique features of the data collection mechanisms required for distributed system development.

The user interface and the database design also reflect this sense of generality. Note that the database relies on a small number of basic types of building blocks – objects, entities, relationships, and behaviors. These generic building blocks are then used to construct a representation of an experiment. In the case of the Concept Demonstration system, this experiment data consisted primarily of entity data file names to be used in executing the SIM DRIVER software.

Object-Oriented User Interface

The preceding remarks alluded to the second observation made about the accomplishments of the DSME effort – the generality of the user interface. The user interface provides a set of basic building blocks that supports the state of the art in windowing, pull-down menus, etc., while providing an interface for the user to insert information into a generalized database. The key to the generality of the interface is actually found in the database structure with which the user interfaces.

There are four basic building blocks that can be used to construct the required data organization: objects, attributes, relationships, and scripts (which define the behavior that applies to the object). In the Concept Demonstration system, scripts were not utilized, although the capability to define them was identified in the analysis phase, and thus was implemented as part of the User Interface software component. In the DSME Concept Demonstration system, the objects are experiments and data files. Attributes of data files include their location within the system. The most significant aspect of the relationship element is that certain data files are used in certain experiments. These concepts are expanded in the overall DSME concept to include experiments, experiment groupings, system abstractions, and so on.

The generality achieved through this approach is clearly attributable to the object orientation of the database structure, and by extension, to the user interface itself. Certainly the ability to instantiate a version of the DSME database based on this object orientation is one of the most significant concepts that can be demonstrated in the Concept Demonstration system. The value of Ada as a programming language that supports the development of object-oriented systems is also evident in the User Interface software component.

The User Interface software itself actually has little direct coupling to the nature of the data entered into the database. One can easily conceive of very different applications of the User Interface. The application of interest in this discussion, however, is extension of the user interface and database structures to support experiment definition for other phases of the distributed system development phases, such as analytic experiments or analysis of performance data collected from operational systems.

There is another extension with potential application within the DSME context as well. The User Interface component provides a generalized approach to defining experiments for any type of simulation, not just simulations involving distributed systems. In the case of SIM DRIVER, the simulation application was actually that of an air surveillance network. However, the user interface implemented on the current effort could provide a baseline capability for a generic simulation experiment user interface that would provide some commonality for users who must deal with a wide variety of models and simulations that have different user interfaces.

Distributed System Performance Data Analysis

One of the exercises performed in the early part of the analysis involved determining the types of data to be collected and analyzed as part of the distributed system development process. The results showed that there is virtually no difference between the performance data collected for a single processor operating alone and the data collected for a processor working in conjunction with others as a distributed system. The same basic performance measures apply. The major difference in analyzing distributed system performance is primarily the manner in which the data is interpreted and analyzed. In particular, the analysis of distributed system performance involves looking for linkages between the behavior of each processor. However, at the level of the actual data collection, the same data requirements exist.

This is a significant observation in the DSME context, since it follows that one can begin the population of the environment with tools that collect performance data from individual processors and individual interprocess communication links. There are a number of such tools, ranging from operating-system-level monitoring calls to more sophisticated performance analysis packages. Development within the DSME context can then emphasize issues such as developing more sophisticated database structures (for defining experiments, abstractions, etc.) and developing tools for data analysis to help the user explore the causal links within the collected performance data that lead to a better appreciation of the distributed system performance.

Data Collection Mechanisms

One issue that was never adequately resolved in the current effort was the optimum approach to data collection. As discussed in detail in Section 4.2.2, there are several methods that can be applied to collect data from a System Under Study. In general, the tradeoff is access to more data versus the intrusiveness of the data collection method, which translates into the amount of work necessary for incorporating a System Under Study into the DSME environment.

For the Concept Demonstration system, the approach was to define "data probes" that accessed data items in the global memory of the SIM DRIVER system, formatted the data as necessary, and wrote it out to a binary file containing the data. The data was easily obtained, due primarily to two aspects of the SIM DRIVER system. First, in the DGTS portion of SIM DRIVER, output messages can be defined in the data header portion of a module, so messages could be easily defined to contain the data required by a particular probe. Second, the output mechanism for messages was similar to that required for probes, and therefore few structural changes to SIM DRIVER were required. A third factor, which should not be overlooked, is that as prime contractor for developing the SIM DRIVER software and the DGTS predecessor contracts, PGSC was intimately familiar with the structure of the software and, therefore, could implement the probes with a minimum of learning time.

Some reservations exist about the generalizability of the approach implemented in the Concept Demonstration system. The approach, which involved inserting probes that capture and output data, is an intrusive one. It was necessary to insert lines of code in the functional models of the SIM DRIVER software to perform the data capture. It was also necessary to modify the executive portion of the software to handle the scheduling of turn-on and turn-off times for the data capture flags. It is not clear how difficult these tasks could be for software that is not as well suited to the insertion of probes, or for "legacy" software that is developed by someone else.

It should not be perceived that the preceding discussion invalidates the general tradeoff discussion in Section 4.2.2. There is still a severe limitation in terms of the data that can be collected without making modifications to the System Under Study. There may not be any reasonable way to eliminate the requirement to make some modifications. Perhaps the best

course of action is to use a method by which certain information (ideally extracted from the symbol table of the System Under Study software) can be stored in the database and subsequently accessed when data collection requirements are defined as well as when actual data collection is performed. This approach would still require some means of storing detailed information about the storage scheme used by a System Under Study, which could possibly require manual input of the data. The Concept Demonstration system demonstrated that the probe concept was viable in certain cases; whether it is generally the best choice remains an outstanding issue.

Commercial DBMS and Statistics Packages

The final observation made about the results of this effort involves the usefulness of commercial off-the-shelf (COTS) systems for relational database management and statistical data manipulation. For the Concept Demonstration system, the RIM database management system was used. RIM demonstrated that a relational DBMS was appropriate; however, since RIM was obtained at no cost, it does not provide the full spectrum of services and capabilities that a COTS relational DBMS typically provides. Thus, procuring a COTS relational DBMS and substituting it for RIM is one enhancement that could be made easily to the Concept Demonstration capability.

The same basic discussion applies to the software for statistical analysis of the data. For the Concept Demonstration system, there was no software available that could reasonably be called statistical analysis software. RIM report generation tools were used to provide a prototype reporting capability. A wide variety of COTS statistical packages provide the functionality required for the DSME environment. As with the relational DBMS, a quick (though not necessarily inexpensive) enhancement to the Concept Demonstration system would be the addition of a COTS statistical package that could be used to perform much more sophisticated analyses of the data captured and could be stored in the performance database.

6.2 FUTURE DIRECTIONS

This concluding section outlines recommendations about future work that can build on the results of this effort and further develop the DSME concepts. Six steps have been identified as part of the progressive development of the DSME capability in its ultimate implementation. Note that these steps immediately follow from the results of this contract -- that is, any and all of these steps could be undertaken immediately. These six steps are:

1. Further define the relationship between the DSME and the DISE.
2. Develop instrumentation tools.
3. Further develop the baseline DSME data collection mechanisms.
4. Insert more powerful relational database management systems and statistics packages.
5. Develop the concept of abstraction to provide generality.
6. Conduct a test on another System Under Study.

The results of these studies should then be applied to the overall development of the DSME

concept, following a traditional sequence of design and implementation. The following five subsections address the steps listed above in more detail.

6.2.1 DSME/DISE Relationship Definition

An RADC activity that was concurrent with the DSME effort was the definition of the Distributed System Evaluation Environment, known as the DISE. The DISE is an ongoing RADC/COT initiative to provide an environment in which researchers can investigate and demonstrate issues of concern for a distributed operating system, such as fault tolerance, reconfiguration, reconstitution, etc. The eventual plan is to expand the DISE to be an environment "where true distributed applications can be written, experimented upon, and verified."¹ Goals for the DISE, as outlined in RADC-TM-87-5, include the following:

1. Provide an environment for both the design and development of distributed systems and the demonstration of systems integration technology.
2. Provide researchers with the tools necessary to evaluate current distributed system technology.
3. Provide Air Force command and control systems with the capability for "system evolution."

As indicated in this report, there is clearly a relationship between the DSME concept outlined here and the DISE. However, as the concepts evolved somewhat independently and simultaneously, the precise nature of that relationship has never been fully defined. There are a number of potential relationships. The DISE could encompass all aspects of the DSME concept. If so, DSME could exist as a separate component of the DISE, or perhaps the DSME functions could be incorporated into the DISE individually. This approach makes sense by virtue of the fact that the DISE is much further along in development than DSME. In fact, there is an existing set of hardware which functions as the DISE. On the other hand, the DISE could be part of the overall DSME concept, fulfilling some of the functions defined for the DSME. The concepts defined for the overall DSME in Section 2 address a broad spectrum of activities, perhaps even broader than those defined for the DISE. A third alternative is that the DISE and DSME could be cooperative, each providing their own unique contributions to the distributed system development process.

A reasonable first step beyond the current state of DSME would be to conduct a study specifically to address this issue. Such a study would have the following objectives:

1. Define the relationship between the DISE and the DSME.

1. Newton, Anthony M., and Sweed, Richard H., "Distributed System Evaluation Environment, An Introductory Report," Rome Air Development Center, RADC-TM-87-5, April 1987.

2. Define the role of DSME in the future distributed system development activities at RADC/COTD.

Note that in this section the groundwork is being laid for addressing the latter issue as well. However, a significant component of the overall future role of DSME is based on the first issue, the relationship of the DSME and the DISE.

6.2.2 *Develop Instrumentation Tools*

The data collection probes implemented in the DSME Concept Demonstration system represent an extremely limited set of possible data collection and instrumentation possibilities for the DSME concept. Another logical step to build on the current DSME work would be the development of more instrumentation tools.

Given the overall objectives of DSME, the emphasis in the selection of these tools should be on the computer system-performance data collection, rather than on functional performance (which was emphasized in the DSME Concept Demonstration system). The rationale for this recommendation is twofold. First, some significant issues remain to be addressed and resolved in the area of performance data abstraction (see Section 6.2.4). Secondly, instrumentation for system performance will provide a more general set of tools, which will allow more rapid development of the potential Systems Under Study that could benefit from the DSME environment.

The tools should be interfaced with the DSME User Interface. The User Interface was intended to provide a general capability to support the user's insertion of information into the DSME database. Clearly the necessary information concerning the instrumentation tools should be part of the DSME database. Thus, the instrumentation tools should be interfaced with that database and invoked via user inputs entered using the User Interface.

One issue that should be addressed as part of the development of instrumentation tools is identification of the specific tools to be included. Some initial thoughts on the types of tools have been documented in this report. However, neither a definitive set of tools nor an appropriate priority for the implementation/installation of such tools has been established.

6.2.3 *Enhanced Data Collection Facilities*

Several improvements to the system service data collection implementation would be beneficial to the DSME. These involve at least the partial automation of the functions normally associated with a programmer familiar with the SUS. Automating such functions would reduce the time and potential errors normally associated with them and, in addition, could permit a non-programmer to accomplish at least some of them.

Automatic Collection Statement Insertion: One requirement of the current DSME approach is that data collection statements be inserted in simulation code to implement new collection requirements. Although this may not occur often, the task is typically subject to error, even if a highly skilled programmer familiar with the SUS accomplishes the task. At

least two functions are normally associated with this goal: verification of input statements and reformatting/recompilation of the SUS.

This function has potential for being system-assisted as described in the following scenario. A special editor (perhaps programmed in DEC TPU) permits an analyst to browse through the SUS source as desired. When insertion of a storage statement is desired, a menu selection from available statement types inserts a template, which is completed with additional menu selections for parameters based on the template and, perhaps, enhanced with contextual information.

Automatic Collection Module Insertion: The mapping that serves the communications between the SUS and DSME software could also be used by software to generate data collection procedures. The mapping includes the specification of data types and their relation to the distributed system data abstraction, providing sufficient information to generate a standard storage routine in a specified statement. Clever use of Ada generic capabilities could be utilized in such a scheme.

While some specific enhancements can be made to the system service approach, there is also a need to reconsider the direct database approach. An analogy can be made to the implementation of the direct database approach and the operation of a symbolic debugger: direct access to memory is available via the mapping provided by the language and linker symbol tables. The mechanisms provided by a symbolic debugging tool are general in access, but specific to an operating system environment. Such a mechanism could perhaps be duplicated and enhanced for accomplishing the same data access for simulations. This general, direct mechanism could provide access to any SUS data element at any time, with only limited interference. A more thorough evaluation of the costs and benefits of such an approach must be conducted prior to pursuing a development goal, but the potential benefits are extensive.

6.2.4 COTS DBMS and Statistics Packages

The quickest approach to adding functionality to the DSME Concept Demonstration system is the addition of commercial off-the-shelf (COTS) software packages to fulfill the functions of database management and statistical analysis. Note that this is not necessarily the most inexpensive approach, since a fully functional relational DBMS for mid-range systems can cost in the range of \$150,000 to \$200,000. Statistical analysis packages, while not typically that costly, could still add substantially to the cost of this step.

Despite the expense, the payoff for taking this step is quite high. One observation of the DSME Concept Demonstration system is the value and importance of the database management and statistical analysis functions. The database management functions in the DSME Concept Demonstration system were implemented using the RIM-5 DBMS. RIM-5 was selected since it is in the public domain and therefore could be used at no expense. That provided the opportunity to develop the remaining portions of the DSME Concept Demonstration system around a relational database concept without the expense of

procuring a COTS system. Facilitated by the standard relational database manipulation operations, the User Interface was easily able to store and retrieve the data required to exercise the DSME Concept Demonstration system. SQL-type statements were used as the mechanism for entering and retrieving data in the database, which justified the decision to use a relational database scheme.

While the RIM-5 DBMS provides the basic functionality for interfacing with a relational database, it lacks many of the features that have evolved over the past few years and that are now standard with commercially available relational DBMSs.

The purchase of a COTS relational DBMS would provide this additional functionality with minimal change to the DSME Concept Demonstration system. The only significant changes would be modifications to the syntax of the SQL statements generated by the User Interface for entering and extracting data from the database.

Another advantage of a COTS relational DBMS is that it provides a standard interface. If the DSME provides a standard DBMS for subscriber simulations, preferably by providing a simulation interface to a commercial DBMS, several distinct advantages are realized. First, new simulation systems need not include data management routines, thus speeding their design and implementation. Secondly, the standard format of the database permits direct access to the standard analysis components and to other simulations. The issues that must be addressed are primarily oriented toward the performance of the database functions. Speed and storage space must be appropriate for the needs of the simulation and must not restrict its usability.

There are a number of vendors that provide relational DBMSs for DEC systems. For example, the following list identifies products that, as of February 1989, were hosted on both VAX/VMS and Unix systems (Unix is also suggested because it reflects the potential for portability):

- DataFlex
- Empress
- Enterprise
- Informix
- Ingres
- Interbase
- ORACLE
- Prelude
- Progress
- Recital
- ShareBase
- Sir/DBMS
- Supra
- Sybase

- Unify 2000.²

As the list shows, a wide variety of products should be evaluated for inclusion in subsequent versions of DSME.

The situation for statistical analysis is nearly identical to the DBMS situation. In the DSME Concept Demonstration system, no statistical analysis functions were implemented. Because there are existing packages that can ultimately fulfill the analysis functions, no relevant concepts would have been demonstrated by implementing some statistical functions. Note that a COTS DBMS may have some of the necessary analysis functions as options for retrieving data. However, more sophisticated analysis functions, particularly involving statistical and hypothesis testing, are likely to be found only in packages that contain functions specifically oriented toward statistical analysis.

Because of the fact that no statistical functions were implemented, a statistical package would provide a greater enhancement to the DSME Concept Demonstration system than a relational DBMS would. It would allow the running of more interesting tests and produce more interesting conclusions using the Concept Demonstration system.

Potential COTS statistical packages that are hosted on both VAX/VMS and Unix systems include, but are not limited to:³

- BMDP
- IMSL Libraries
- Minitab
- P-Stat
- RS/QCAII
- SPSS.

In addition to purchasing relational DBMS and statistical analysis software, some simple enhancements can be made to the DSME Concept Demonstration system that would substantially enhance capabilities. For example, since DSME provides a common user interface, it is only natural to use it to control experiment execution. Limited experiment control will be provided in the current DSME, but a more complete set of functions could be standardized and included in the system. The first potential extension would be the provision of data modification within the executing SUS. Other control mechanisms might include a generalized restart capability.

Since data collection is accomplished by DSME or DSME-related software, it is reasonable to permit the redirection of such data to a monitor function that processes and displays selected data elements. For the current effort, it is expected that simulation time will be the only data displayed.

2. "RDBMS Market Sizzles with New Products," *Digital News*, 6 February 1989.

3. "Choose the Right Statistics," *Digital News*, 5 February 1990.

6.2.5 Develop Concept of Abstraction

One of the key ideas that surfaced during the analysis phase was the notion of a formalized abstraction of system performance that could be used as a framework for defining experiments, data collection requirements, and measures of effectiveness. As described in Section 4.2.3, an abstraction is a set of fundamental entities, attributes, and relationships included in all instances of a class of systems.

For example, at one point in the development of the DSME Concept Demonstration system, consideration was given to including an additional entry in the DSME database that represented abstractions of C³I systems. The concept was that any model of a C³I system would include entities that mapped to the defined abstraction of the fundamental entities defined in the abstraction. New simulation models of C³I systems could be added to the DSME environment by defining the mapping from the specific entities in the model and the entities in the abstraction. Since experiments could be defined on abstractions, it would then be possible to insert a new model and conduct the same basic experiment, using a new model and/or a new C³I system, without redefining the experiment. The concept of the abstraction would also facilitate the comparison of data from different experiments dealing with different systems.

Although the concept of abstractions was not implemented in the DSME database, the basic idea of the abstraction is still an idea worth pursuing. However, a number of issues that must be addressed were left unresolved in the development of the DSME Concept Demonstration system.

One issue is the difficulty in determining, for a class of systems, which entities, attributes, and relationships are sufficiently fundamental for inclusion in an abstraction. Part of the problem under the current effort probably is due to the fact that C³I systems are among the most difficult to categorize and among the most difficult for which to develop a taxonomy. Beyond the simplistic concept of a Collect Information/Make Decision cycle, an abstraction of a C³I system tends to fall apart among the many varieties and variations of C³I systems that exist today. Furthermore, there is very little in the way of a science of C³I systems on which to base a taxonomy or abstraction.

The concept of abstraction would be more easily demonstrated with a class of systems that has a more inherently fundamental structure and in which there is a more scientific basis for considering the systems. One example would be computer processing systems. An abstraction would deal with fundamental entities such as processors, memory, registers, busses, secondary storage devices, controllers, and so on. The abstraction information could then be entered into the DSME database, and experiments could then be defined against that abstraction.

Another problem encountered in the definition of abstractions involved an overall schema for defining the abstractions in the DSME database. An attempt was made to define the schema, but was abandoned when it became apparent that there was insufficient

time available to derive a meaningful abstraction of C³I systems.

In considering the utility of abstractions, it is useful to note that at least some relationship exists between the data collection and data analysis processes. In fact, the initial attempt at defining data collection requirements for an experiment could easily involve the selection of experiment (substitute analysis) goals, which are then translated into data collection classes and elements. Although such a relationship is already provided by the intersecting hierarchies, traversal from the more detailed levels of the trees to the more abstract levels is not a direct process, thus limiting the transition from abstract analysis nodes to abstract collection nodes. Another direct mapping should exist, permitting the association of such data classes as communications objects with experiment goals such as "evaluate communications bottlenecks."

Given the preceding discussion, a logical next step in the development of the DSME concept would be additional work on developing the concept of abstractions. A useful test of the evolving concept would be to develop abstractions for at least two classes of systems. One class should be computer systems, since they are relatively well understood. The other class should be one that is easier to deal with than C³I systems, although the results of the study should be applied to C³I systems as well. Part of the study would include defining a structure within the DSME database for storing the abstraction information; it would also include the actual implementation of the abstraction within the DSME Concept Demonstration system to prove the concept of the abstraction.

The issue of abstraction must also be addressed from the perspective of distributed systems themselves. As detailed in the DSME *Functional Description*, the analysis of distributed systems is based upon concepts that are identical for simulated, prototype, or operational systems. The analysis of distributed systems also provides a specific application for the experimentation abstraction described above. As previously described, there are objects and classes applicable to any distributed system.

What is not yet clear is whether a set of abstract experiment goals can be developed for distributed systems. Such goals could be expressed in the form of questions, or in the more definitive form of MOPs/MOEs. A thorough investigation of these concepts could very likely result in significant progress in the development of distributed systems and also provide significant capability within the DSME.

6.2.6 Another SUS

The final recommendation for the next step in the development of the DSME concept is extension of the DSME Concept Demonstration system to include another System Under Study (SUS). There is substantial value in applying the concepts demonstrated thus far to a different SUS. All the design decisions made to support the generality requirement can only be evaluated by implementing a different SUS.

It should be noted that the original intent of the DSME effort was to do precisely that – implement the DSME system for both a tactical model and a strategic model. For several reasons this implementation strategy was dropped under the current contract, but should be reconsidered as a next step in the DSME evolution. Note that a strategic simulation need not be the other SUS. Any simulation with some relevance to RADC/COTD would make a credible candidate.

One candidate SUS is the Distributed System Simulator (DSS), currently being enhanced by Harris Corporation under the RADC/COTD Internettted Systems Modeling (ISM) effort. Other candidates for integration include DeNet, a distributed system simulation system under development at the University of Massachusetts at Amherst. Additional study is required to evaluate the potential benefits of such simulations within the DSME and the costs associated with their integration into the environment.

6.2.7 Further System Development

The preceding subsections all deal with steps that could actually be pursued immediately from the existing DSME Concept Demonstration system baseline. This reports concludes with some comments on what could lie beyond specifically oriented studies.

Once the five individual steps are taken, a great deal more information will be available on which to base the remainder of the DSME evolution. The step beyond that would be a system design effort addressing the DSME concept in its entirety, along with an implementation plan and road map to show how incremental development and changes to the DSME Concept Demonstration system could instantiate that design.

The ultimate result would then be a system that meets the requirements outlined in Sections 2 and 3, created through an incremental and evolutionary development sequence. This system would make a significant contribution to the development of distributed systems.



MISSION of Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C³I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C³I systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.